CANONICAL SIMPLIFICATION OF NETWORKING AND THE INTERNET

Submitted in partial fulfillment of the requirements for

the degree of

MASTER OF SCIENCE

Department of Computer Science

Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

New York, New York

by

V. GURUPRASAD

08 FEBRUARY 2005

“Mister, sir, Mister – a humble M.R.C.S. ... A dabbler in science, Mr. Holmes, a picker up of shells...”

Dr. James Mortimer, responding to being called “Doctor”,
in *The Hound of the Baskervilles*, Sir Arthur Conan Doyle, 1901.

Every technology goest through three stages: first, a crudely simple and quite unsatisfactory gadget; second, an enormously complicated group of gadgets designed to overcome the shortcomings of the original and achieving thereby somewhat satisfactory performance through extremely complex compromise; third, a final proper design therefrom. ... Despite their mad shortcomings, these “automobiles” were the most characteristic form of wealth and the most cherished possessions of their times. Three whole generations were slaves to them.

Robert A. Heinlein, in an otherwise fictional work *The Rolling Stones*, 1952.

Diya talé andhéra. (The base of a lamp lies [well hidden] in [its own] shadow.)

Old Hindi proverb that well describes the *representational principle*.

ABSTRACT

Canonical simplification of networking and the Internet

V. Guruprasad

Introduced are two novel schemes for inter-networking, and an opportunity they present to *diagonalize* the Internet architecture, i.e. orthogonalize its components at multiple levels, so as to make it simpler as well as inherently more general, dynamic and scalable. The first is client-side virtualization of Internet Protocol (IP) addresses, representing a functional inverse of network address translation (NAT) that instantly enables unlimited *effective* extension of Layer 3 address space as independent realms, analogous to per-process virtual addressing in Unix, thus also eliminating the current need for global coordination of the Layer 3 space. The second is a namespace providing IP-like routing semantics instead of mere translation to lower layer addresses, sufficing for inter-realm addressing and routing independently of Layer 3. It is further shown to be a *natural coordinate system* by construction, thus obviating the express numbering of nodes as in IP, and *canonical* with respect to networking, in the sense of requiring the least configurational information of any networking, i.e. addressing and routing, scheme. These properties make it ideal as an *inter-domain network and protocol*, and for confining IP to individual domains or realms using VAS.

Simplicity results for network operators by the elimination of all need to coordinate IP addresses, including for application servers, requiring only locally unique labelling of nodes and link-local configuration. Generality includes full multi-realm access to unmodified IP hosts and applications – via local VAS mapping of foreign destinations addressed by name. Flexibility lies in the capability for multiple application-specific secondary namespaces and for *bottom-up* evolution of newer inter-networks even over existing infrastructure by linking separate deployments, as coordinated numbering is eliminated. The dynamic nature includes the instant effectiveness of name bindings and deletions. Scalability is assured by the elimination of hard limits and generally by the *localization* of both configuration and traffic. Route discovery and automatic subscriptions to namespace changes are envisaged for performance and efficiency, and filesystem-like ownership and access control as a simpler security model. The basic ideas are implemented in a prototype.

ACKNOWLEDGEMENTS

I owe the privilege of getting into a Masters program for a second time, this time into computer science and at one of the best schools in the US, to IBM. More particularly,

- to C. J. Tan and the Deep Blue team, of which chance saw fit to make me a part, during the time when the initial ideas of this work were taking shape;
- to my IBM managers, Gabriel Silberman and Murray S. Campbell, and several colleagues, notably Ronald Mraz, A. Joseph Hoane, Matthew Thoennes and Daniel Oblinger, for their invaluable encouragement and academic guidance;
- and to R. J. T. Morris, our director at that time, for this mandatory surprising academic gift.

Working at IBM Research served to elevate ideals and standards to where they should be, in contrast to my prior life in India for three decades, that expressly forbade world-class achievement in any form or field, forcing me to pray not for a second chance in life, but a first chance to live. It took an oracle, however, specifically the then Science Advisor at the US Embassy in New Delhi, just as the 1991 Gulf War began, and some extraordinary friends, to make that possible. IBM sheltered me precisely from "green card" to citizenship, and even a fantasy of learning martial arts under Shaolin monks has come to pass – careful what you wish for, says my *shrfu*, lest wishes be granted! I can think of no simpler explanation than that the gods granting them are real and human, the ones that created the world and the opportunity that enables a life apart from mere existence. And of no better way to thank than to further wish – may the ideas presented here become real and profitable for all!

My desire for a Unix-like, mature operating system approach to handling Internet configuration and use, as will be described in Chapter 1, would have never matured into a thesis had it not been for critical pointers and advice from Dinesh Verma of IBM and Jonathan S. Shapiro, now with the EROS project (`http://www.eros.org`), that eventually led me to discover the canonical construction. Numerous other colleagues and peers at various workshops, conferences and in the IETF, most notably Brian E. Carpenter, have helped me crucially in understanding the field, as acknowledged more specifically in the journal paper [Gur03]. The paper included a general treatment of route discovery for the canonical scheme and thus presented a complete networking scheme in a more conventional sense. It was still unsatisfactory, and meant two more years of groping for a clear and succinct solution that adequately addressed the large scale naming, routing, multicasting and mobility problems.

This comprehensive picture has emerged only over the past year or so, with much needed validation and support from the NIRA [Yan03] and FARA [CBFP03] papers at the FDNA 2003 Workshop, the first of which gives much needed validation and support for the mostly hierarchical routing envisaged in the canonical approach. A second key idea, that of derivative links to map between application-specific hierarchies and the base canonical topology, had been conceived in late 2002 as a possible solution to the naming and namespace issues pointed out by several IETFers, notably Keith Moore and John Stracke. However, only during final assembly of this thesis in recent months did its suitability for addressing the remaining three problems, viz of the large scale naming, mobility and multihoming, become clear, and as explained in Chapter 1, it critically rests on the vast experience of the authors of the Nimrod and FARA proposals. Several general aspects of this scheme have also paralleled, if not predated, the TRIAD project at Stanford [CG00], which has thus also been of considerable value as reaffirmation of similar viewpoints.

Since 2003, I have had the honour of working with the R&D of System Management Arts, Inc. This has provided me with at least a second hand feel of the actual problems and practices in the management of some of the largest provider networks in the world. Further, the Smarts InCharge$^{\text{TM}}$ products demonstrate immense scalability in root cause analysis because of their model-driven top-down approach, fully complementing the top-down construction of networks themselves as advocated in this thesis.

I describe next in the Preface that this thesis seems to fulfill an initial expectation of mine, viz to demonstrate a deceptively simple principle of "representational closure" that promises to replace the current postulative foundations of physics with a formal system of reasoning purely from definitions. My emerging results in this area rest on the farther vision made possible by famously giant shoulders.

I must thank Columbia University itself, and my advisor Jason Nieh, for inspiring and demanding the highest standards, and waiting patiently for both the ideas and my technical articulation skills to mature to this point. I am sure it will take substantially more work to make this approach a practical and common reality, and it is my sincere hope that both will be supportive of a larger effort to this end.

Also, my pursuit of the MS program and the course credits it entailed would have been impossibly difficult without the help of the staff of the Columbia Video Network (CVN), from the first course all the way to the completion of this thesis. Lastly, my use of the first person, instead of the "royal we" frequently preferred in the literature today, is an acknowledgement that any and all of the ideas presented here could be hopelessly wrong, and it would me, and my often limited ability to understand mature perspectives and advice, that should stand to blame.

V. Guruprasad
2005.02.02

PREFACE

It is difficult to design a whole new networking architecture, even when one is working in the field and has made important contributions therein, like the pioneers of IPNL [FG01], notably Paul Francis, and of TRIAD [CG00], David Cheriton. It took Francis almost a decade of hunches, as shown by PIP [Fra94], to arrive at IPNL. To work outside of a field, alone and unchartered, and yet contribute on architectural scale borders on the insane, or, as one of my colleagues put it, asserting an egalitarian right to contribute. Luckily, the difficulties have been mostly of my own learning.

My group at IBM was not involved in network or Internet architecture, but with embedded high speed networking and deep computing, notably of chess. The present architectural quest began with my shock, the first time I had the occasion to look at socket programming in 1994-1995, at the mainframe-like primevality of IP addressing, as expressed in Fig. 1.4. It is interesting how many opportunities emerge if one looks across from one's own field of expertise to another – around that time, I also had to explain to an examiner at the US Patent Office how my electromagnetic heat engines, which scale to elementary particles instead of the huge, inefficient mechanical pistons of emerging nano-technology (IEEE Semitherm XIV paper, 1998), overcame a basic defect in Edison's generator of 1892. I have yet to take on the egalitarian challenge to convince the commercial automotive and power generation worlds.

In the present case, the gap has been narrower, but all the more demanding. The key problem, tantamount to a canonical, self-contained approach to network address space construction, had been well identified for me by my learned colleagues, and had been solved in summer of 1999 with the core "addressless networking" principle. It is gratifying that three major ideas, NIRA [Yan03], FARA [CBFP03] and DHTs [BLR$^+$04], have emerged in the past year and half, providing authoritative support.

Another aspect of the present work that deserves mention as a deeper motivation, is my hope that it will help lead to a greater appreciation of the information sciences as more than a matter of mundane engineering, and instead as the formal discipline for reasoning about the very knowledge of scientists, embodied by actual, if as yet ununderstood, representation in their brains. The fundamental difference this makes is that one can dispense with the current axiomatic foundations of both relativity and quantum mechanics. The axiomatic approach has become increasingly fashionable in the last century, leading to the well known electro-weak unification (Salam, Weinberg and Glashow, Nobel Prize 1979) and the modern string theories, but these unifications are almost purely mathematical, and *fundamentally incapable of providing a deeper understanding of physics* than the postulates. My approach would be to make science *model-independent*, exploiting formal techniques of computing.

The problem is that the postulates, such as those of special relativity, irreducibly *predicate* natural behaviour, instead of merely identifying irreducible patterns from which the behaviour should be inferrable. In the appendix of `gr-qc/0005014` on the archive server at `http://www.arxiv.org`, I have been able to show that the speed of light postulates can be inferred from no more than precise notions of measurement, calibration and the basic notions of what qualifies as "fundamental". This derivation was in fact appreciated by a leading astrophysicist colleague at IBM, and the article *per se* was cited by NASA in their 2002 report on the anomalous acceleration detected by the Pioneer missions, but has not been published for want, till lately, of substantial terrestrial consequences to justify changing a widely accepted perspective.

A related development is a complete classical derivation of Planck's law, proving that the Planck's constant h is merely the equivalent of Boltzmann's constant k_B for the Fourier transform (frequency) domain. This reveals a further problem with the axiomatic approach – that it stifled inquiry into the very assumptions of Planck that broke away, in hindsight fallaciously, from classical physics and electromagnetics. This result would have been impossible without first hand involvement in thermodynamics, rediscovering the exploratory first steps of Watt and Carnot without the cloud of later knowledge, by inventing a fundamentally new class of heat engines that overcome at least two fundamental limitations of all mechanical engines, including nano-engines: enabling conversion of individual hot particle energies in parallel, like single-(piston) input-multiple-(work) output (SIMD) in computing, and competing against diffusion, instead of waiting for the hottest temperatures to disperse to lower grade. The Planck result has in turn provided indispensable insight in a very recent, more practical result exploiting these deeper insights of wave behaviour, notably providing a dramatically new way to separate overlapping signal spectra from multiple sources by exploiting distance information contained in their phase envelopes, *without regard to the signal modulation and content*, i.e. independently of FDM, TDM and CDMA[1].

Common to all of these results is the *principle of representational closure*, restated in Section 1.6, that *observable physics must be fundamentally limited or constrained by the properties of the physical representation of the observed variables in the observers and instruments.* The problem this principle revealed for relativity is that all observed numerical values of physical variables are ratios of observed physical variables to the calibrated markings on the physical instruments, which, together with the physical referents of scale used for their calibration, would be "covarying" with the observer, but the local referents are unrepresented in both Einstein's and current formulations of relativity theory, and unmentioned in the entire relativity literature. When their covariation was considered, the constancy of the speed of light and the implication that it is the maximum speed of information not only quickly emerged as consequences,

1. Paper to be presented at IEEE WCNC 2005, New Orleans, titled "Bandwidth relaxation by Space Division Multiplexing". While it has been demonstrated with some acoustic samples using Octave (and Matlab), the method is still largely theoretical, and a simulator is being developed.

but accompanied a simple yet accurate prediction of the (apparent) acceleration of the universe and the (apparent) dark matter/energy ratio, almost two years before their discovery in 1998. This prior insight eventually led to the uncovering of a systematic error in the calibration procedure being used in all of our deep space telescopes, which inadvertently falsifies the digital data to make the "Hubble constant" uniform across all observations[2], and to the signal separation method mentioned. It has also led me to uncover complementary shortcomings in current physics, such as a complete lack of treatment of high order diffractive losses applicable even to visible light on the cosmological scale, and to neutrinos within dense matter, suggesting a fundamentally different philosophical answer to Olbers' paradox and the appearance of the universe – current astrophysics and quantum field theories are limited to Fraunhofer and Fresnel diffraction effects, which are mathematically defined only for cumulative diffraction angles less than $\pi/2$ and consequently lead to fundamentally wrong notions of lossless ($1/r^2$) propagation in the not-so-free matter bearing universe. Recent doubts on the validation of inflation theory by the WMAP space observatory data seem to suggest that the microwave background is less than cosmological in origin, and could even be consistent with my local diffractive back-scattering theory!

The deeper motivation within the present work can now be explained. As explained in Section 5.3, the coordination-free, self-addressing namespace technique I describe in this thesis is also a natural coordinate system for networking, exploiting the links of a spanning tree simultaneously as a logical representation of the spatial separation between the nodes. To assert my egalitarian right in physics, I needed an illustrative application of the representational principle outside of pure science and specifically in computing, yet concerning physical space. Hence networking and particularly the fundamental issue of addressing therein, relating to coordinates.

This background of physics should not be interpreted as a divergence of interest, however, but rather the converse: in the rest of nature, when one starts with a point focus, one is forced to diverge – there is always comfort of immediate acceptance and job security in extrapolation, but that only leads to ritualistic thinking as in inflation theory and IPv6! Fundamental developments, which is what research should be about at least some of the time, requires to be like wavelets starting from different points on the previous wavefront to progress concurrently to newer points of focus, to quote directly from Huygen's principle.

There is much scope for further work, notably the simulation of route discovery and network evolution. The approach might even gain wider acceptance more quickly if recast as a mundane but more elegant application name space tool, say as "a self-coordinating universal TCP grid", and to let the users discover for themselves, if at all, that they no longer depend on IP. This has been a temptation, and also suggested by Carpenter, but it might not have made a thesis!

-vg

2. Detailed as supplementary material in US patent application 10/884,343, filed 2004.07.02.

CONTENTS

LIST OF FIGURES

NUMBERED LISTS

LIST OF THEOREMS AND PRINCIPLES

CHAPTER 1

PROBLEM, PERSPECTIVE AND MOTIVATION

This thesis treats network *addressing* as a problem, as opposed to the usual view in the Internet Engineering Task Force (IETF), summarized by the rhetorical question "what can be simpler than assigning (numeric) addresses (and routing with them)?" [Lixia Zhang, 49th IETF meeting, San Diego, 2000]. The answer developed here, with other architectural implications, is "*letting the computers do it!*".

Almost all of networking research has assumed *a priori* assignment of network, or Internet Protocol (IP), addresses. Only a small body of work addresses the problem of addressing itself, and is so far limited to the Dynamic Host Control Protocol (DHCP) and related protocols for the lowest level subnets, and some similar work in configuring remote routers. The difference lies in where the boundary between configuration and automation is drawn. Current working notions of *names*, *addresses* and *routes* [Sho78] place this boundary (broken line, Fig. 1.1 bottom) *after* the addresses, thus allowing for the automation only of the routes through routing protocols.

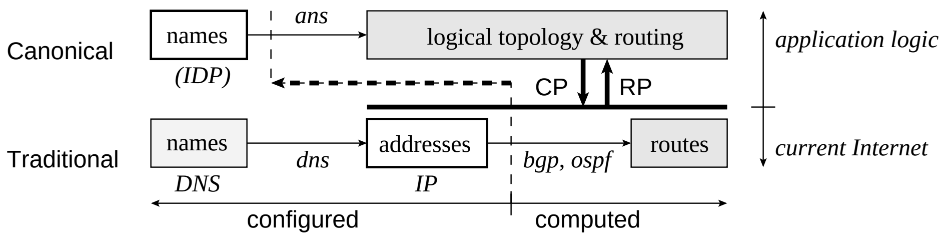

Fig. 1.1: The configuration-automation boundary

This thesis establishes a canonical principle (CP) of networking, by arguments of the most general needs of applications and users, that this boundary can be shifted left (broken arrow), *enabling addresses to be also automatically managed.* Simplicity, flexibility, responsiveness and scalability are improved, illustrating a representational principle (RP), inspired by high level languages and operating systems (OSs), that *names are by definition sufficient for representing all application entities.*

The remainder of this chapter explains

- that this change was as such overdue as a new logical topology layer (see figure) *to correct an architectural defect of the current Internet* (Section 1.1), as well as for usability and self-operating features similar to those of host OSs that were necessary for the proliferation of computers today (Section 1.4);
- that it would yield the usability and automation benefits like those of host OSs and would come from similar architectural design (Section 1.5);
- that it primarily concerns operation and usage, and is hence inherently general, not only "backward compatible" with existing, unmodified IP applications and hosts, but also providing them full extended (name-based addressing) capability (Section 1.2), as opposed to Internet Protocol version 6 (IPv6) and alternatives hitherto proposed, which generally require protocol stack changes (Section 1.3);
- that the change constitutes a lightweight limit for several ideas being explored for future routing, operational support, multihoming and mobility (Section 1.3);
- and that the limit fundamentally represents the spatial separation of nodes, and is therefore *canonical* for networking, so that any other scheme would be more wasteful, overconfigured, restrictive in design and less scalable (Section 1.3).

The present treatment will establish

- that the canonical construction provides *a natural coordinate system*, enabling network nodes to be identified and default-routed to by their positions without express numbering and with no route computation;
- that the approach *eliminates a perceptional problem* that had *needlessly limited IP routing to a single address space*, necessitating IPv6 and related ideas, and requiring protocol stack changes and address translation gateways therein; and
- that the architectural correction concerns *an orthogonalization of roles* for want of which numerous protocols in the IP suite have had to reinvent inter-domain authentication and negotiation and the handling of firewall issues.

1.1 Need for an inter-domain protocol (IDP)

Names are undeniably more appropriate than number spaces for humans to deal with, but the traditional view of names as merely symbolic references to addresses implied an impractical level of automation to make names the primary means of addressing. The view treats names as a flat space, permitting no information of network locality or topology in the names themselves. The Domain Name System (DNS) endorses this view in order to separate naming from routing. This however places IP addresses in the role of identifiers, in network administration and in many applications.

The penalty is not just the known burden of manually administering the IP address space. It also includes general unresponsiveness and inflexibility consistent with over-configuration, as will become clear, and complication of transport protocols as well as security, since inter-domain routes must be negotiated across firewalls for want of end to end address visibility at Layer 3 ([HS01, Hai00]), which cannot include built-in support for the high level semantics necessary for security and authentication.

This is an architectural flaw, since, but for our knowledge of Internet evolution, we would expect an *inter-network* architecture to concern bridging *administratively independent* networks, which means dealing with matters of ownership and security. These should have been the issues of concern since the first firewall, even before the IPv4 address depletion issue and the invention of network address translation (NAT) [Ege94]. The Internet architects, IETF and the networking community at large have failed to duly recognize that an evolutionary shift has occurred, from needing to link *dissimilar* infrastructure networks within a *single, trusted academic domain* of the old Arpanet to connecting *similar* [Internet Protocol version 4 (IPv4)] networks denoting *independent proprietary domains.* A corresponding shift in architectural perspective means would be to recognize IP as an efficient and effective *intranet protocol* and to seek a distinct *inter-domain protocol (IDP).* The IDP must provide global if coarse addressing *and* support application level topology semantics, but *an intranet protocol needs to do neither* (which IP does very well!).

Both these roles are currently left to Layer 3 in the Open Systems Interconnections (OSI) reference model (cf. [Tan96,§1.4]), inevitably leading to case-by-case discovery of problems with firewalls, as evident in [HS01, Hai00, Car00], and reinvention of solutions for firewalls and security, i.e. IDP problems, in numerous protocols, including the file transfer protocol (FTP), the hypertext transport protocol (HTTP), the real time streaming protocol (RTSP), the Session Initiation Protocol (SIP), the Simple Object Access Protocol (SOAP), etc. as well as application programming interface (API) frameworks like the Sun Java 2 platform, Enterprise Edition (J2EE) family.

The need for explicit IDP functionality in routing seems to be long felt, as apparent in the Forwarding directive, Association and Rendezvous Architecture (FARA) scheme [CBFP03]. Other similar ideas form the basis of a new Internet routing architecture (NIRA) [Yan03]. However, these schemes are still too conventional and betray lack of recognition of the IDP issues: neither considers naming at all and they both continue to assume global Layer 3 (numeric) address spaces, which, by definition, would need to be coordinated and would again leave security and authentication to afterthought.

This thesis presents a new approach in which inter-domain routing of connection requests and datagrams occurs fundamentally at the application level, via a high level network of names constructed as a canonical coordinate system. The application level addressing of datagrams has an illustrative significance concerning current difficulties of relaying connectionless protocols, as will become clear. The approach would further simplify the configuration and usage of IP networks substantially due to the removal of global coordination constraints on configuration and the high level form of addressing, respectively, and both relate to host OS evolution and design, as will be explained.

1.2 Sufficiency and generality of names

This thesis will more particularly establish the sufficiency of names *carrying only user and application semantics* as a high level form of identifiers for defining *logical network topologies*, independent of and free from any form of addressing defined, managed or controlled by the network infrastructures. The idea is to decouple applications from infrastructure nuances by use of the IDP, and, conversely, the infrastructure operators from application constraints like backward compatibility, which has been a significant part of the difficulties of IPv6 migration, for example. The decoupling would also yield an infinite *effective* address space, and subsume “6to4” as well as relaying between IP and non IP networks, while allowing equal access from unmodified IPv4 hosts and networks, and simplifying network configuration and operation.

The first benefit, infinite effective addressing, reflects the high level name form and generality required of the IDP. The generalized relaying benefit follows as, with IP addresses no longer expected to route across domains, the IP end sockets would be interfaced to IDP *asymmetrically* at each end of an application connection. The very notion of an IDP to transport messages and negotiate connections across unrelated networks means that it could perform these functions across dissimilar IP networks, non IP infrastructures like native asynchronous transfer mode (ATM) or multiprotocol label switching (MPLS) clouds, and even with non IP end points. Even the ongoing IPv6 migration would become unnecessary in terms of its primary motivations, viz a large enough address space to accommodate next generation telephony and similar applications, and the anticipated return to Layer 3 transparency [Car00]. Proposed alternatives to IPv6 envisage a larger network of independent IPv4 address spaces, called *realms*. The IDP subsumes the role of inter-domain addressing, while permitting a simpler and more powerful design, as will be demonstrated below. The simplification alone is significant, sufficient to compel adoption even without the promise of a larger effective address space, as well as necessary to enable such a larger network, since the current architecture poses impractical complications, well illustrated by the arduous evolution of IPv6 design and migration efforts.

The configurational and operational simplification follow from the elimination of the global coordination requirement of a single Layer 3 space, since each realm can be managed individually by itself, without regard to one another, with all requirements of inter-realm coordination deferred to the IDP. Further, it is anticipated that the notion of realms will itself become reduced, from their current treatment as replicas of the current Internet to the autonomous systems (ASs) and individual corporate or private networks, as a result of this thesis, since each such network as such constitutes an administrative and ownership domain.

The expected simplification will be shown to be substantially more than a simple isolation of the domain address spaces. The IDP would need to be in the form of an *overlay*, in order to be decoupled from both IP and non IP realm infrastructures, and provide *name-like absolute addresses* that could be used like DNS names by existing IP hosts and applications. This rules out the non routing, database-like design of the DNS, although it will be shown, as an intermediate step of reasoning (Section 5.7), that the DNS could be overloaded, in its current design, with inter-realm routing.

An alternative namespace will be described with the remarkable property that it is *a natural coordinate system* not needing address or route configuration, and being name-like, is suitable for immediate use by applications and users, as well as ideal, as will be established, for application specific logical topologies and user controllable routes. This contrasts with the flat character of DNS and IP spaces, and the IP-like form of inter-domain route selection envisaged in NIRA with no application-definable features. It will be further shown that such logical topologies would be highly dynamic and scalable, notably in the addressing of application servers and in contrast to the relatively static nature of IP. It thus fulfills FARA's goals but with newer usability features, quicker response to change, no global coordination, and better scalability.

Fig. 1.1 depicts this duality of simultaneously enabling extremely dynamic logical topologies (RP), and simplifying the administration and operation (CP). While CP concerns downward automation of address and route management within domains, RP implies sufficiency of the higher levels to represent all possible low level topologies, and therefore for managing them at both domain and multidomain (or Internet) levels. The RP essentially questions any need for configured non routing numeric identifiers, as seems to be envisaged, for instance, in FARA.

The notion of a *naming overlay* acting as a primary network address space in its own right, without global coordination at the lower layers, and doing so more simply, efficiently and scalably at the same time, seems unthinkable from the traditional ideas of networking. *It is intuitive in virtually every other field*, however: in an OS, a single filesystem name space is used across multiple, separately formatted disk drives, and in militaries around the world, generals, not messengers, coordinate armies. These ideas of IDP decoupling and natural name-based coordinates are adequately formalized by the following principles, which are demonstrated by detailed examples in Chapter 3, and justified in Chapter 5 using basic properties of the name tree structures.

Principle 1 (Unlimited inter-networking). *A necessary and sufficient condition for unlimited effective addressing using fixed length numeric addresses is that inter-realm routing be done via names.*

Principle 2 (Canonical principle of networking). *The least configuration needed for efficient network addressing and routing on any scale comprises local link interfaces and locally unique node names.*

The result could be described as a *diagonalization* of the Internet, as it represents orthogonalization at multiple levels. The rest of this chapter is architectural discussion – one may skip to Chapter 3 first for a basic description of the present approach.

1.3 Related work

The inherent overlay design of the IDP, which is the key contribution of the present thesis, makes it simpler than both IPv6 and alternative schemes, notably the Nimrod [CCS96], which is reincarnated in FARA [CBFP03]; IP Next Level (IPNL) [FG01], which extends NAT for multirealm routing; and TRIAD [CG00], since each of these requires modifying the IP address format and adding corresponding "shims" to the host protocol stacks for the inter-domain routing. Each further envisages critical use of the DNS or a very similar namespace on the inter-realm scale, thereby at least partly overloading it with inter-realm routing, which would still be handled at Layer 3. The configurational complexity would thus grow linearly with the number of realms and an import/export scheme would be also required for accessing destinations in foreign realms, as discussed in IPNL [*op. cit.*]. The complexity can in no case be *less* than in the existing Internet, which is promised exclusively by the present approach.

FARA, on the other hand, is a *meta-architecture* for mobility and multihoming in a single addressing realm, and calls for an efficient scalable means for translating static IP-like numeric end point identifiers to routable numeric addresses to be dynamically set up through provider negotiations. The essence of FARA is thus to separate the identifier and routing roles of IP addresses; its danger is that the first is less functional than the second. Unless the routing address assignments can be efficiently and fully automated, one ends up managing two sets of numbers instead of one (Fig. 1.2).

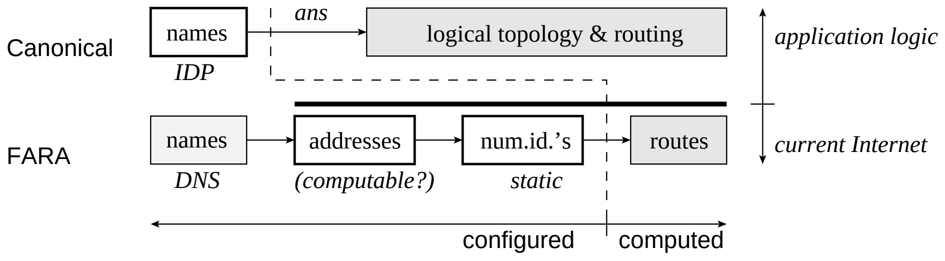

Fig. 1.2: Comparison with FARA

It is as such debatable whether there is a substantial need for non routing numeric identifiers. They are admittedly useless within the static infrastructure and intended only for end points, but their only possible role in setting up routing addresses would be to replace Media Access Control (MAC) device ids or user identifiers, which should depend on higher level ids and cryptographic signatures. FARA's real objectives are a generalization of Mobile IP and for multihoming, but reflecting conventional wisdom, which is misplaced as will be demonstrated by the alternative namespace of this thesis, that translation of names to non static routable addresses would be impractical.

This current wisdom rests on the prevalent view of namespaces as databases as in the DNS, recent variations for layered dereferencing consistent with FARA [BLR$^+$04] and cooperative, decentralized operation [RS04], and in service-oriented distributed databases such as Sun's Jini, the Session Description Protocol (SDP) [HJ98], and the Intentional Networking System (INS) [AWSBL00]. In retrospect, the current view is equivalent in a host OS, as will be discussed in Section (1.4), to requiring application processes to not only share a single virtual address space[1] but also the symbol table!

This prevailing view is contradicted by the key design, in the present approach, of a very lightweight and dynamic namespace that *interprets* hierarchical, filesystem-like pathnames directly like IP addresses for routing and signalling, instead of looking them up to return numeric addresses [Gur00]. No translation occurs in the use of this lightweight namespace, as the hierarchical relations embody network links directly, and the destination is reached by following them, like inodes in a filesystem to navigate from one directory location to another. It uses the maximum possible distribution of

1. E.g., as in IBM's AIX, which at least up to version 3.2.5, had a 53-bit "single paging space" virtual memory, within which every variable of every process could in principle be directly addressed.

name labels: one per node, as if it were a collision-free distributed hash table (DHT), to compare with the layered dereferencing scheme [*op. cit.*], but no global coordination is needed for set up, as each link is configured using at most local information. It not only satisfies FARA as an extreme DHT, but would obsolete it and the IP tradition of coordinated identifier spaces altogether.

As an *everywhere-locally* set up system with little need for replicating information, it epitomizes the limit not only of lightness, but also of responsiveness to change, as only the immediate link needs to be created or changed to effect the addition, deletion or relocation of an entire subtree, and furthermore, such changes take effect instantly. These properties would be diluted, as will be shortly described, by use of caches and the corresponding need for cache invalidation or updates. However, the caching would be purely for incremental improvement in performance not critical to operation as in the DNS, and with little overhead to justify remaining with coordinated addresses.

NIRA proposes similar, mostly-hierarchical inter-domain routing, but it concerns Layer 3 routing, thereby continuing to depend on a coordinated Layer 3 address space and placing itself in conflict with the current use of BGP and OSPF routing protocols. The overlay-IDP form of the present approach uniquely empowers its deployment as a "super name-address" space over the current infrastructure as is, and *to make full use of the latter.* NIRA cannot operate other than by replacing the Border Gateway Protocol (BGP) just like IPv6 cannot penetrate core routing without displacing IPv4, but the requirements for an IDP outlined in Section 1.1 include sufficient generality to optimally exploit all such infrastructure nuances and features.

Another key idea in NIRA is user access to and control of inter-domain routing, and NIRA's critical contribution lies in showing that this can be realized along with the routing policies of the ASs as currently handled by BGP. User control of inter-domain routing had been anticipated in the present work in the form of *guided routing* by the namespace topology as first described [Gur00], and *lateral route tables (LRTs)*, populated by route discovery protocols [Gur03], were subsequently incorporated as means to overcome the triangular inefficiency of hierarchical routing [Gur02a]. As the LRT entries would be representable as $dest\_name \rightarrow next\_node\_name$ mappings, a NIRA-like facility for selecting routes would be not only straightforward to implement, but can be intelligently used by lay end users, instead of just network administrators or skilled users. User control of inter-domain routing is intended, in NIRA, to promote user choice and fairness of revenue passed on to the core networks, and both purposes would be better served by wider accessibility to this feature allowed by the name-like form in the present approach.

NIRA more specifically proposes hierarchically guided discovery, which would not yield the full topology but would be especially efficient on inter-domain scale. While the idea appears appropriate to and can be easily adopted over the present IDP, there is opportunity for subscribed route updates (SRUs) to deliver IDP topology changes to the LRTs, as all such changes would be initiated manually or by applications.

SRUs would be the preferred means for maintaining *derived links*, which appear to be a more accessible generalization of FARA's notion of identifier-address separation allowing multiple levels of administration and indirection. In particular, they would enable provider-independent and application-specific IDP topologies over a base IDP network that includes provider and user network gateway identifiers in the hierarchical path names, and would be otherwise worse than IP with respect to mobility and multihoming, and worse than the DNS with respect to identifier-route separation. However, these "virtual namespaces", with their LRTs, not only suffice for the non topological naming like the equivalent of `www.ibm.com`, but would be more dynamic than ever possible with any database approach, including the existing DNS [JSBM01] and the proposed cooperative, decentralized form [RS04]. Indeed, although the latter is designed to be self-organizing, the feature only refers to how it would organize the configuration information, not whether that information could be as such reduced. Other database schemes proposed with this feature, like content addressible networks (CANs) [RFH$^+$01], fare no better in this regard.

Another current "meta-architectural" idea that sounds somewhat related is that of a Knowledge Plane (KP) over the Internet [CPRW03], which would mostly formalize operating support systems (OSSs) used in major networks. Like NIRA's hierarchical discovery and user route selection features, this too would be presumably more easy to deploy over the present IDP, but otherwise unrelated – administrative tools similarly exist in host OSs, for example, but are not considered intrinsic to the OS itself.

Large scale auto-configuration of IP addresses using cryptographic techniques to efficiently synthesize and map unique identifiers to routes [For03] would appear to be similarly motivated. Optimal structuring of such networks has also been described in *ad hoc* networking literature [KRS02]. The basic problem with both approaches is that they seek to replace the *definition* of routable destinations by automation – *it is because IP addresses define destinations that they need to be manually coordinated today.* The *ad hoc* approaches cannot succeed on the large scale until this definitional functionality can be efficiently fulfilled differently, and this is precisely what is done in the present approach by recognizing, via the representational principle (RP), that the destinations are only meaningful if defined and used by applications and users. The observation further means, however, that *synthetic* identifiers can be only useful for automatic processes, and not for the businesses of human users. The canonical theory will show that the cryptographic approach is overly complicated even for this role, and it will be shown that the optimal structuring problem too admits dramatic simplifications without the constraints of static IP addressing (Sections 6.4, 6.5).

Other auto-configuration mechanisms are also similarly constrained: "zeroconf" protocols to link-local networks; DHCP, with optional use of dynamic DNS, to lowest router subnets; and various *ad hoc* networking schemes to small or special networks, as mentioned. The present approach eliminates these constraints and would be simpler in each of these scenarios. Another interesting class of ideas that has yet to make a visible difference is that of *Active Networks* such as ANTS [TW96, Wea99] from MIT. The present approach, by virtue of its OS, filesystem-like high level form, should make it a powerful enabling platform for application-level active networks (Section 1.5).

1.4 Following the OS evolution

As stated, the present approach appears to be a necessary next step in the evolution of the Internet, emulating the evolution of key host OS features like process virtual memory address spaces and filesystems. Internet Engineering Notes (IENs) discussing the current operating notions of names, addresses and routes [Sho78, Coh78a, Coh78b, CC78] reveal that memory addressing terms from host system architectures of the time were considered, as well as name-like addressing used by the postal services, before understandably settling on the fixed length numeric form for IP. Considerations of variable length addressing have generally failed as the efforts tend to devolve into the fixed length form for logistical reasons [Dee00], as in the case of CLNP (cf. [Per99]).

All of these past considerations, however, including IPNL and FARA, are *bottom-up* in their design perspective, requiring the coordination and configuration of a global Layer 3 address space, over which all functions, including application level identifiers, are implemented. In a modern host OS, however, the main, comparable coordination is at the command and application level, with the memory and disk sector allocations being performed automatically and on the fly to satisfy higher level needs at process and filesystem levels, respectively. A coarse convention, e.g. using high order memory for the OS kernel virtual memory and root inodes or "superblocks" on the first and last disk sectors, is all that now remains at the lower level.

This was not always the case: precursors to the mainframes lacked virtual memory and sported only assemblers to translate symbolic names to memory addresses. The address blocks were naturally reused between programs, as with the private addresses of NAT, which makes the current Layer 3 approach even more primitive, as mentioned in Section 1.3. Current complacence with this situation is due to viewing networking as a special circumstance that does not ordinarily present a similar need for separate "renumbering" in each application. This need occurs in distributed parallel computing for addressing peer end points – the message passing interface (MPI) standard [MPI97] provides for a multitude of shared *communicators*, each comprising an ordered set of peer end point descriptors, and thereby acting as a virtual address space. Such needs are adequately fulfilled by user-space translation to IP, however.

Filesystems happen to be a relatively modern feature. The early mainframe and personal computer (PC) OSs like CP/M provided only one "directory file" that listed files by sector numbers; the sector allocations had to be managed manually. This is similar to the manual coordination of IP address blocks by ICANN and administrators in each of the ASs and user networks, and the special circumstance justification of the single (virtual) address space for IP has thus entailed extensive manual involvement. This not only explains the slowness of the generally desired IPv6 migration, but also accounts for much of the delay in deploying new applications, including Voice-over-IP (VoIP) in enterprise networks (cf. [Rea03]).

The per-process virtual memory addressing and filesystem features of modern OSs are crucially important in terms of usability, as they insulate uses and applications from physical memory limitations and the management of system memory and disk storage, respectively, providing applications with rich abstractions of these functions. These abstractions generally do not diminish accessibility to the low level features of the implementation, but instead promote better use as well as control of the host resources, with at most pathological exceptions that are not of general interest. What makes these abstractions doubly significant to the present context is that *they also lead to freedom for automatic management of these resources in the first place.*

An analogous OS-like layer for automatically managing very large networks and the Internet would be substantially more desirable if it provided similar abstractions to improve usability and control, which is indeed the case in the present approach, as illustrated by the case of NIRA-like inter-domain routing explained in Section 1.3. The OS analogy also implies that these abstractions would be necessary to allow such management, by decoupling the resource protocol interactions from their high-level representation, as mentioned in Section 1.1.

Fig. 1.3 illustrates how this new layer would fit as a natural next step of progress. It follows the tradition of the famous end to end arguments (EEA) for separation between networking and end system functions [SRC84], that led to an effective merging of functions across Layers 5, 6 and 7 in IP [Tan96,pp.35-44], by arguing for a similar, but mostly conceptual, merger of Layers 4 (transport $\sim$ TCP,UDP), 3 (network, IP), and 2 (datalink), as IP addresses can have only localized significance under an IDP, as explained in Section 1.1. In the figure, the circled numbers denote design generations, and a comparable layer model of a modern host OS is also shown.

The further argument that the current Internet architecture lags behind OSs in terms of comparable usability and control features is illustrated by the time-line plot of the evolution of such features in both OS and networking fields in Fig. 1.4. The figure suggests that networking has lagged behind the comparable host OS developments not merely by a decade-long skew, but by an entire generation, as IP is both operationally

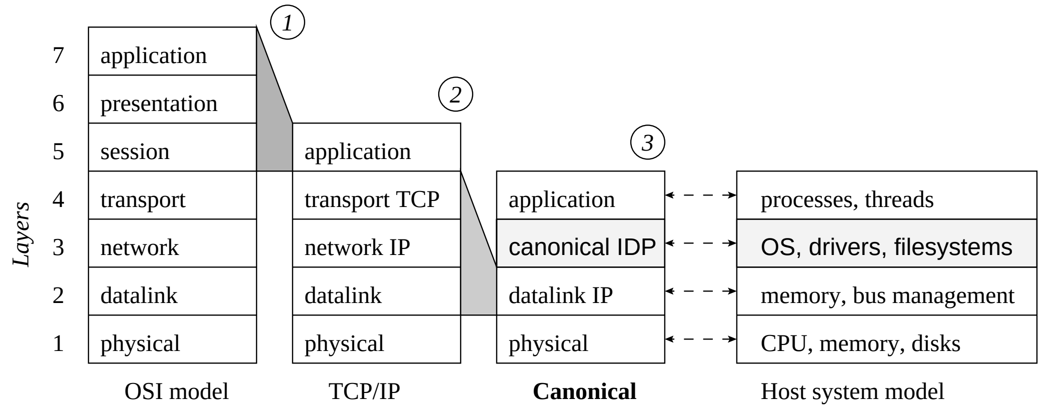

Fig. 1.3: From OSI model to OS-like layer

and by design closer to the mainframe generation of 1960s than to Unix®and related OSs from the 1980s over which much of the IP suite was born.

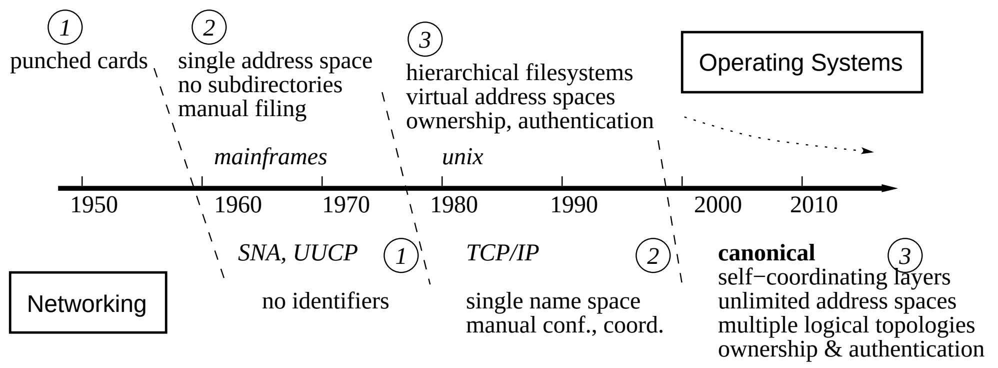

Fig. 1.4: The OS evolution parallel

Not surprisingly, these features are also missing in the other schemes discussed in Section 1.3. TRIAD [CG00] has been exceptional in similarly identifying names as a necessary future direction, but it has not called for an IDP and offers no efficient, scalable name-based addressing solution of its own suitable for this role and for multi-realm routing of connectionless protocols.

Indeed, Unix itself, as well as versions of Microsoft's Windows®containing both features as such emerged only after computers became more widely accessible, and the Internet has become comparably accessible only as of the late 1990s. The second generation effort leading to IP has involved, as in the OS and processor evolutions, wider participation of engineers, and the previous proposed alternatives to IPv6 reflect this well engineered approach. However, Unix also required a complete rethinking in terms of usability, starting with the Multics project, notably for office tools.

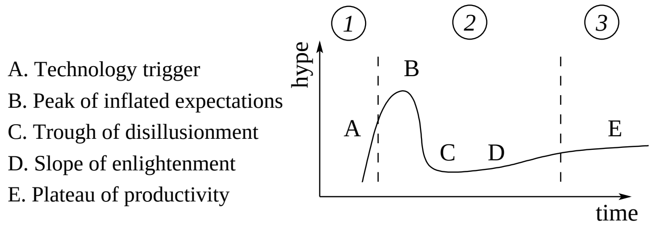

Fig. 1.5: Gartner's hype cycle

The generation numbers shown in the above figures also correspond to Heinlein's famous description of the stages of technology, quoted in the opening pages of this thesis. In particular, the third generation OS features also seem to be signatures of technological maturation likely to stay for a long time. Gartner's "Hype Cycle" diagram, Fig. 1.5, which was inspired by the Internet boom of the late 1990s, suggests interpretation consistent with the present call for an IDP layer. As stated, however, the proposed changes would be largely perceptional because of the generality and the overlay form required for an IDP, as explained in Section 1.1.

1.5 OS-like usability, security and automation

The need for an IDP layer has been especially unobvious, as remarked, as it conflicts with the traditional ideas represented by the OSI model. In particular, it is unintuitive that an overlay can, let alone should, provide a global high level address space and that too without global coordination of the identifiers. These aspects are precisely what enable OS-like features and functionality, as described below in terms of filesystem-like namespaces, virtual addressing and greater generality.

The only well known use of names for network routing prior to TRIAD, outside of postal services, has been Unix-to-Unix Copy (UUCP) for email, and the main defect in the scheme was the absence of absolute, path-independent destination identifiers. UUCP accordingly faded when such addresses became available with IP.

In the present approach, absolute, i.e. source-independent, identifiers are obtained from a canonical construction of links using only link-local set up information. A first layer primary space of such identifiers would be left intentionally route-identifying, in the sense of identifying routes along the namespace links, but it would still remain source-independent and therefore absolute, though only partially path-independent. Path-independent identifiers are then obtained by setting up secondary spaces over this route-identifying first space, similar to how the DNS currently operates over IP.

The first layer decouples the infrastructure networks from applications sufficiently to permit the desired OS-like self-management capability. It also yields a hierarchical name space including provider network nodes and AS gateway identifiers as a higher level administrative interface convenient for both infrastructure management and for defining the secondary namespace entries. The coordination of non topological names is thus cheaply obtained at the secondary name space level without the inconvenience or limitations of numeric addressing.

For example, the non topological brand-name site `www.ibm.com` would be treated, in the filesystem paradigm to be shortly further explained, as the path `/com/ibm/www`, and configured as a derived link for `/us/att/ny/armonk1/ibmhq5/lan22/aixhost3` say. This explicitly reveals the provider hierarchy, so that if the provider were changed or the site hosting otherwise moved, the derived link would need to be updated. This can be automatically done if, say, an IDP automatically subscribes to its target IDP for change notifications concerning the mapped target. The actual mechanisms would be of less interest than the fact that subscription and notification are a known method for using less traffic than polling (under applicable conditions; see Section 6.6).

Clearly, a variant of DNS A-records could be defined for mapping non topological identifiers directly to the primary name space, and the (IP) infrastructure would still be equally decoupled. However, we would be then stuck with sensitivity of the first layer space to the provider chains, which is likely to be more than in IP and to require more frequent tracking updates. We would have to pay a penalty in DNS operations for the privilege of decoupling IP. This appears to be the case with the DHT variants of the DNS being researched as well (see Section 1.3).

The manual updates could be avoided using of dynamic DNS updates, but that would call for additional, albeit static, configuration. Indeed, any database approach, including the DNS and its variants, would be more complex, if not also slower and less wieldy, than the present canonical construction, as the former invariably depend on copying the name bindings to caches across the network for operability on significant scales. Caching is envisaged in the present approach, but less critically in the form of LRTs, as stated in Section 1.3, whereas the (existing) DNS would entail two almost end-to-end network traversals (to the server-side authoritative name server and back) just to fetch the address, in absence of caching. In any case, the canonical approach has the basic advantage of a much simpler setup.

Further, the canonical construction requires only a reliable link mechanism, and can therefore be executed over existing IP networks using TCP, qualifying it as an overlay. The overlay property means that multiple and possibly overlapping spaces, both primary and secondary, can be operated over a given infrastructure. One could similarly envisage multiple, concurrent instances of the DNS, but their configuration and operation costs would be prohibitive for their limited functionality. The present IDP is not only simpler but would simplify the deploying IP networks by eliminating the need to coordinate their address management.

The present IDP would in addition enable filesystem-like usage including security and authentication functions. As a first step in exploring this, an Internet FileSystem (INFS) scheme was designed to reuse Unix filesystem directory cache and navigation kernel code for hierarchical network name spaces, and was discussed on the IETF mailing list in the course of this work, along with open source release of Linux kernel module sources doing this for the DNS [Gur02b], for comments. The DNS module suffices to illustrate directory-style navigation for network names, enabling usage of the form `cd /inet/com/ibm/www; w3render < tcp:80`, etc., in addition to smaller "software footprint" ($\approx$ 22 kB), fewer context switches and related performance gains.

The responses have been consistent with the novelty of the idea, the closest relative evidently being the BSD portal filesystem [SP95, McN99], which enables indirect conversion of DNS name strings to open sockets by linking up the file `open(2)` system call to a conventional user space resolver. It does not use the kernel directory cache, however, and cannot provide similar navigation. With the primary name space layer of the present IDP, the navigation can be made more functional, as the filesystem-like pathname in the primary naming layer would include the provider chain, each node of which can be associated with a security domain and authentication requirements, permitting automatic configuration of layers of firewalls. As will be briefly discussed later, this is a first step towards full auto-configuration of large IP networks.

More immediate applications could be use of filesystem *read/write/execute* notions for networked content, local directory listing (*ls*) to complement the directory-style navigation, and analogous ownership and access control. The first two would represent a host OS-like form for Active Networks. The last concerns automatic configuration of firewalls: as the first layer name space hierarchy would be as such aligned with the provider chain, the interior network at each node is as such correctly identified by its immediate child links and does not need to be specified. This would simplify security and access control methods so much that, if combined with distributed authentication, the ownership and management of subnets in an enterprise could be delegated to the actual users and managers without network management skills, as with directories in the Andrew filesystem (AFS) or IBM Transarc's Distributed File System (DFS). An extreme possibility briefly examined in the course of this work is that of automating even the first time Layer 3 configuration on enterprise scales, i.e. "bootstrapping" the IP configuration along with security and firewalls dictated by a central recipe. Such a capability would be important for recovery in case of a large scale shutdown, and is possible so long as the node identities are individually preconfigured.

It would be anticipated that an overlay IDP could provide relaying for packets at Layer 3 for IP end points or Layer 2 for direct non IP end points, and the availability of a cheap, efficient and scalable IDP would make global coordination of addressing at these lower layers redundant and avoidable. The main use of a Layer 3 (IP) network would then be to connect across dissimilar Layer 2 networks to form uniform, efficient *intranets*, as explained in Section 1.1. This would eliminate a global Layer 3 address space, but global network access would remain fully provided via the IDP, so at each node, the Layer 3 packet addresses must route to the next relay gateway.

We would nevertheless want Layer 3 addresses to be consistently associable with specific destinations at least during the relay sessions. This means that, at each node, the instantaneous view of the global, multirealm network as embodied by the Layer 3 addresses must define a consistent address space relative to that node and possibly its realm. This dynamically changing view would closely resemble the Unix notion of the virtual address space of a process, as shown in Fig. 1.6, in which the $127/8 \equiv 127.x.x.x$

reserved "loopback subnet" is shown being used as the host-relative virtual address pool for relayed foreign destinations, and is accordingly called an (asymmetric) virtual address space (VAS). The indicated mapping does not require any change to the IP routing in the deploying network, nor a compromise of access to any existing Internet destination, and has therefore been the one used in the prototype demonstrations.

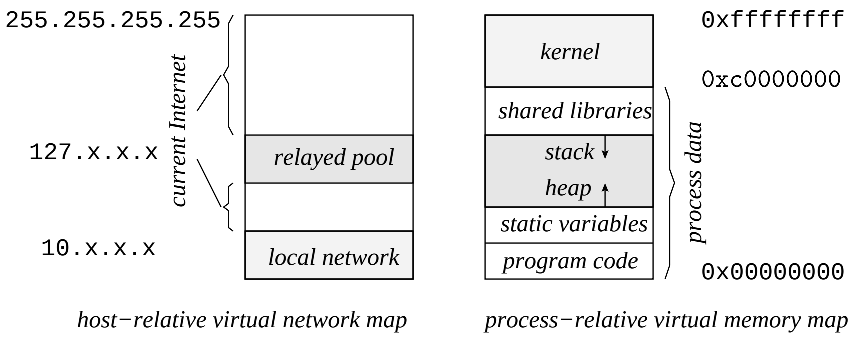

Fig. 1.6: Similarity of VAS to Unix virtual memory

The only known scheme for virtualizing IP addresses is the Address Virtualization Enablement Service (AVES) which uses the DNS as the negotiating IDP, but done at the server side, using a large network of relaying "waypoints" to extend the virtualized connections to the clients [NZS01]. The reason for virtualizing server addresses is of course that in the current Internet, servers are the most sought destinations, and accordingly need to have static addresses, hence their virtualization should have led to substantial flexibility. Instead, the scheme calls for additionally hosting and managing a waypoint network for at most a marginal extension of the IP address space in return.

Specifically, AVES is simply a server-side NAT, and the marginal extension comes from enabling similar reuse of private address blocks. The virtualization proposed in this thesis is of the client side, as just explained, and is a functional inverse of NAT, as will be explained. It will be further established that this inversion is *necessary and sufficient* for unlimited addressing under an IDP.

The only logical constraint for VAS is that the server be visible at least by name, so that such a mapping can be dynamically negotiated. The IDP satisfies this name visibility directly, without *a priori* Layer 3 visibility, and provides the signalling for negotiation and map setup, so that the server and client side networks do not need to share their Layer 3 address spaces, and each can be separately managed without regard to how the addresses are assigned in the other. In fact, each deploying network can make use of the *entire* 32-bit address space of IPv4, or the full 128-bit IPv6 space for that matter, for itself, without loss of access or connectivity to or from the rest of the Internet, since the IDP can trivially interoperate with the DNS at either end, as will also be described. For this reason, IP applications also do not need to rewritten, and the existing IP clients and servers can both be operated indefinitely, again without loss of accessibility or connectivity. An overlay IDP is thus *exactly* what is, once again, *necessary and sufficient* to enable Unix-like virtual addressing and self-management.

The overlay form is also thus sufficient to permit graded partial deployments, by not having to replace neither the physical IP infrastructure nor its existing operations right from the start. Further, such partial deployments would not require renumbering, not even automated, unlike in IP or any of its proposed alternatives: when such partial deployments need to be linked, all that it would take is to link the root of one as a child in another to connect them pairwise into an inter-(domain-)network, or *internet*. In addition, as each of these internets is an overlay, nothing in the architecture prevents concurrent existence and use as application- or demography-specific internets. The same could be done with the DNS in principle, but a DNS is so complex to set up and operate, and takes up so much physical and computational resources, that secondary name services are invariably obtained using other kinds of servers, including databases like LDAP and proprietary implementations. The present IDP is so lightweight that it has an *ad hoc* networking quality of being almost spontaneously set up using slightly less configuration than DHCP on a typical client host.

The main problem with multiple, overlapping IDPs is configuring applications to pick the right one, but so long as a primary, conflict-free IDP name space exists, the problem belongs to application space and is trivial. The only problem with linking up partial deployments is that the IDP addresses in each network linked as child would instantly become longer. The impact belongs at the application level once again, and would be absent in the child's secondary IDPs, for continued local use.

As the virtual mapping scheme is generic, "6to4"-like mappings also do not need to be specially designed or standardized. Instead, as soon as IDP nodes are deployed along a network's own perimeter and the interior DNS configured to default to the IDP, it becomes *instantly free* to manage its interior "realm address space" any way it pleases, even if operating "legacy IP" clients and servers, so that both NAT and AVES are obviated, respectively, and to reuse the entire IPv4 space, or multiple instances thereof by partitioning itself, thereby also obviating future migration efforts like IPv6.

The IDP and the negotiated virtual address space mappings clearly subsume the functions envisaged in FARA, so that mobility and multihoming capabilities are also assured, via the automatically maintained derived links forming the secondary spaces. As the IDP provides cheap default routing by its own construction, participation in an exterior routing protocol is also not necessary and can be deferred. Further, users and applications everywhere can easily deploy dedicated (secondary) name spaces as logical topologies of their own, so that the IP/DNS architecture is as such surpassed.

The above expectations of improved, OS-like usability, are expected to be more compelling, should the formal arguments of canonical networking prove inadequate in this regard, than the mere extension of the IPv4 address space promised in IPNL and TRIAD, the incremental generalization of Mobile IP ideas proposed in FARA for mobility and multihoming, and the naïve restoration of Layer 3 transparency (with full retention of the existing configurational complexity) anticipated from IPv6. They would serve, together with the OS filesystem-like security and authentication model, to substantiate the argument in Section 1.1 that the residual OSI perspective in the current Internet design is indeed an architectural defect, needing to be corrected in future changes to the Internet architecture, as illustrated by this thesis.

An important assumption in these ideas is that *reliable* point to point links are not only possible, but that an extended structure of such links would be stable enough for use as a high level address space. During periods when one or more links are broken, which is likely given that the structure is as such *expected* to be continuously evolving, nodes that are descendents of the broken links will be not only unreachable, but would not exist as addresses. This behaviour is really no different from IP, however, and the descendents of broken link end points could be reachable with their prior addresses via the LRTs. This can be ensured by retaining the LRTs until the links are restored, just like IP route tables are retained between BGP or OSPF updates.

1.6 Principal contributions

The main contributions of this thesis are thus considered to be the following.

A-1. Cognition of a subtle architectural defect in the existing Internet, and of the need for an *inter-domain protocol* to overcome this defect by addressing and routing both datagrams and connection requests *fundamentally* at the application level, i.e. without Layer 3 end to end addressing support, as stated in Section 1.1.

A-2. Principle of a functional inverse of NAT extending the Unix notion of process-relative virtual addressing to client-relative virtual network addressing, thereby enabling unlimited effective Layer 3 address space.

A-3. Principle of derivative, subscription-managed links for supporting DNS-like non topological semantics in the canonical approach.

A-4. Formal treatment of network addressing, yielding the concept and rationale of a *canonical form*. Identification of the canonical construction as a coordinate system, rediscovering DesCartes' notion of coordinate referencing in the context.

A-5. Principle of representational closure, arguing that entities affecting applications would be representable at the application level, and hence that only entities so representable should be relevant to networking.

This is similar in spirit to the principle of observability in quantum mechanics, according to which unobservable quantities should have no place in physics. In the present case, the closure principle holds that connectionless Layer 3 services should be elevated to the IDP or have no place in networking.

A-6. Creation of a new dimension of flexibility, simplification and power, comparable and analogous to modern host OSs, for networking, including enabling of robust OS-like handling of credentials and authentication with Unix process-like address *space* isolation of networks, and filesystem-like formatting and self-management.

The canonical form and virtual addressing have been substantiated by a prototype implementation demonstrating two-way relaying of TCP and ICMP across a firewall to show the generality. Sound reasoning is presented for the rest.

CHAPTER 2

THE ADDRESSLESS ARCHITECTURE

This chapter introduces the overall networking architecture that might be constructed from the present work. It introduces the principal components and briefly describes how they would fit in the overall picture. One could skip to Chapter 3 first for a basic treatment of the key component ideas, and return to this chapter later.

Section 2.1 introduces a key component representing the natural coordinate system by construction. It will be shown in Chapter 5 that the DNS could be employed in its place for the purpose of extending the effective Layer 3 address space to an indefinite number of realms and for some simplification of network operations by administering private networks as independent realms, but the routing namespace would be much simpler to deploy, more dynamic and enables filesystem-like security and general usage features mentioned in Section 1.5. Therefore, while it is possible that a mix of the two will be used for a time, it is important to have a complete architectural picture based only on the new approach. Section 2.2 introduces the remaining principal components of this complete picture, and Section 2.3 explains how they would operate together.

A detailed overview of the rest of this thesis follows in Section 2.4.

2.1 The routing namespace

The key result that started the present thesis is a name-based addressing and routing system [Gur00] that in retrospect constitutes a coordinate system approach, and using only *locally unique* names and *local configuration information* for its construction. It is thus *addressless* according to Shoch [*op. cit.*], and is called the Addressless Networking System (ANS), intended for both primary and secondary IDPs (cf. Section 1.5).

The ANS differs from the DNS mainly by the fact that it routes messages directly to named destinations instead of returning transport layer (IP) addresses to reach them. ANS declarations are themselves links of the logical topology, and *interpreted*, like bit fields of addresses in hierarchical IP routing, by default. It improves over IP in being *name-like* and *coordination-free*, its topology and name-like absolute address space arising from the link configurations instantly *with no nonlocal coordination or computation*. It is thus canonical with respect to networking as global addressing and hierarchical routing are both obtained with neither external input nor computation. This also means that it would allow internetwork topologies to evolve by *a posteriori* linking of partial deployments, instead of by *a priori* assignment of address blocks, in other words, evolutionary construction of internets, as stated in Section 1.5.

The primary and secondary namespace roles would be both handled by the ANS, for reasons of simplicity as well as performance improvements over the DNS and other older, more database-like naming schemes, as already explained in Section 1.5. The primary ANS will be referred to as the *base ANS (bANS)*, since it serves as the base inter-domain address space for the secondary, or *derivative ANSs (dANSs)*. These are the only inter-domain address space components, and, as further stated and remains to be shown for this thesis, require no global coordination.

A dANS's configuration would essentially comprise initial definitions of links to bANS, or other dANS, destinations, and these link definitions would be automatically updated upon bANS changes, using automatic subscriptions to change notifications. These link representations would be very similar to lateral route table (LRT) entries, which are so called as they represent non hierarchical routing. Similar subscription mechanisms could be applied to the updating of LRTs.

One concern with the subscription idea is loss of notifications due to temporarily broken links. This mainly affects LRTs because the only change notification specific to a derived link is that from one end point notifying the other of its own base ANS address change – subsequent propagation of the change would again concern the LRTs of the derived ANS. Notifications arriving while an immediate link is broken could be saved, up to a point, for subsequent forwarding when the link is restored. Whether a general rediscovery is triggered thereafter would depend on how an individual LRTs is set up, because an LRT may be populated by discovery or other means. Especially without LRTs, the ANS routing reflects the locality of its structure, such that higher level ANS servers would be less loaded than comparable DNS nodes.

2.2 Introduction to the remaining components

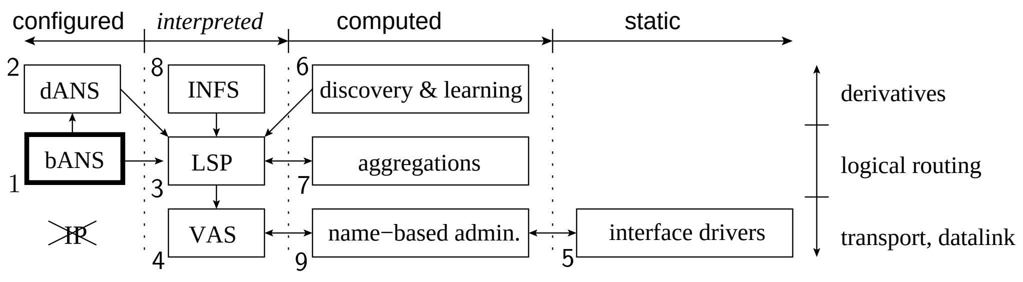

Fig. 2.1: Overview of the architecture

While the ANS suffices for the routing of individual messages, connections and flow routing require the relaying of packets across realm boundaries and firewalls. As the ANS can be used for routing the control messages to set up the relay states, i.e. as the requisite "control plane", there is no need for end to end Layer 3 visibility; label switched paths (LSPs), using labels of local significance, like in MPLS [RVC01] but without a label distribution protocol, are sufficient as the general relay mechanism, and form a third component of the architecture (Fig. 2.1).

A fourth component is the VAS interface to IP or Layer 2 infrastructures and to IP hosts, set up automatically along with the LSPs. The related fifth component is a set of interface/translation drivers specifically ensuring or implementing consistent end to end logical functions as required by the EEA [SRC84] across diverse infrastructure technologies, including reliability for ANS messages. The architecture thus literally provides for separation of end to end functions, including security and inter-domain authentication (Section 1.1), from transport (Layer 4) addressing. The latter, as the Layer 3 (network), should be mainly concerned with the local infrastructure and its utilization, as will be shortly explained. A sixth function also already mentioned is discovery and other possible forms of learning for populating the LRTs.

A seventh functional component depicted concerns aggregation of both LSPs and Layer 3 addresses. For LSPs, a recursive aggregation scheme was developed, to ensure scalability commensurate with the theoretically unlimited addressing assured by the canonical approach. It addresses past issues of the scalability of connection-oriented networking (cf. [Hui00a]), as current IP-based solutions (cf. [FVWB00b, FVWB00a]) cannot be used for inter-realm routing, and should be regarded inappropriate on inter-domain scale in the present view. The scheme exploits the *label stack* of MPLS to aggregate segments of LSPs following the same switches for two or more hops into a tunnel LSP, recursively, so as to reduce the switching table size ideally to a logarithm of the number of connections. The enabling concept is a systolic "parallelism discovery and aggregation" protocol which can be periodically applied in an existing system of LSPs, or incrementally whenever a connection is created or destroyed. In retrospect, its main utility given Moore's law lies in enabling the MPLS framework to dynamically adapt to persistent flow patterns, condensing them automatically to aggregate patterns.

The aggregation of Layer 3 (IP) addresses would be not so much for scalability, as an unlimited Layer 3 effective address space is already assured by ANS and VAS. Rather, address aggregation is an unavoidable operational problem similar to garbage collection in OS memory management, and is constrained by the global coordination requirement in the current Internet architecture. Aggregations need to be periodically performed, mostly manually, by the ASs to release IPv4 address blocks and to thereby help keep the BGP tables manageable in size. As the present approach obviates the coordination requirement and reduces the identifier role of IP addresses, simpler, fully automatic aggregation methods become possible.

An eighth component, INFS, is really an OS and API issue, but becomes important in the present context to represent filesystem-like "look and feel" as embodiment of the representational principle (RP). Likewise, the opportunity for fully name-based administration is shown as a ninth component. This would likely include capability, in the long term, for full auto-configuration of the IP (Layer 3) infrastructure, on enterprise and AS scales, as will be briefly discussed.

2.3 Operating structure and configuration

Fig. 2.2 shows the configured layers in traditional IP and present architectures side by side for comparison. The shaded layers comprise mostly local functions requiring little or no network coordination. This indicated correspondence of roles is approximate, however, as the present approach uses IP only in an intranet or realm role, on par

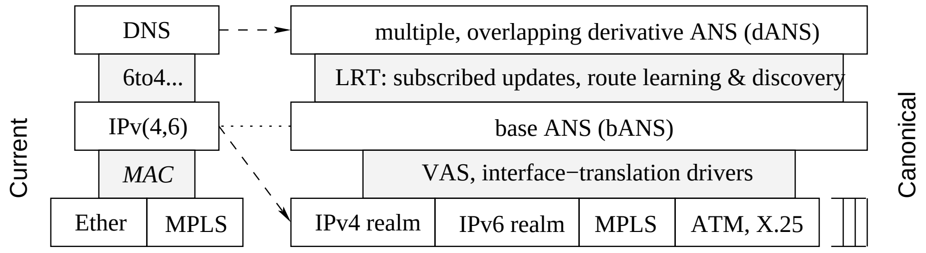
DNS
6to4...
IPv(4,6)
MAC
Ether
MPLS
Current
multiple, overlapping derivative ANS (dANS)
LRT: subscribed updates, route learning & discovery
base ANS (bANS)
VAS, interface−translation drivers
IPv4 realm
IPv6 realm
MPLS
ATM, X.25
Canonical

Fig. 2.2: Inter-network addressing layers

with MPLS, and introduces a new layer, the bANS, in place of IP as the true primary IDP layer. With reference to the host OS model of Fig. 1.3 (on page 14), the bANS and dANSs, along with the LRT and the translation drivers, are the OS-like canonical layer. The figure shows that the greater simplicity and power of addressing achieved can be at least partly attributed to the use of what were hitherto considered as the top layer of network addressing as a less than global foundation layer. Conversely, the comparison serves to further clarify and underscore the need for an IDP with just such a change of roles, as argued for in Section 1.1.

The figure also represents two lesser ideas, the first being that the IDP separation and VAS permit so much asymmetry that, with suitable interface drivers (or “protocol stacks”), both ANS and LSPs connectivity becomes available across dissimilar IP and non IP infrastructure networks. It becomes possible to obtain connections say between an IPv4 end host and an IP-less MPLS destination, even across intermediate networks not enabled for ANS signalling, without specific efforts like “6to4” for IPv6.

The architecture does not preclude continued use of IP for linking dissimilar Layer 2 networks within a given domain, but it also permits IP-less domains, which would be appropriate say for core Internet carriers, as well as unrelated “user realms”, thus leading to an effectively unlimited Layer 3 space and administrative freedom thereof. Its main difference from the OSI model is that the infrastructure technologies can be treated on a par as peers instead of always having to be layered over one another. As explained in Section 1.5, the ANSs would be especially appropriate and adequately robust for this inter-domain addressing role.

The figure also underscores a basic distinction between the derived links forming a dANS and the LRT entries in either layer, both of which map identifiers to lower level addresses. A derived link specifies a remote address and needs to be configured, whereas an LRT entry maps a remote identifier to a local interface or address.

2.4 Overview of this thesis

Chapter 3 explains the basic principles of VAS and ANS, particularly illustrating by examples the construction and use of ANS for establishing connections across both single and multiple realm boundaries.

Chapter 4 discusses the prototype implementation, and the scenarios and operating procedures used in its demonstrations. It brings out the practical details of the ANS construction and usage discussed in Chapter 3, and provides a detailed, packet-level explanation of the present idea of application level IDP for both connection requests and datagrams. The measured connection set up performance is also discussed.

Chapter 5 presents a formal treatment of these constructions, including proof that the ANS is canonical with respect to networking, in both the computational sense of efficient addressing and routing, and in basic sense concerning the very representation of physical space spanned by network links as a coordinate reference frame, hence not requiring any form of numbering, as stated in Chapter 1. Two further aspects of the architecture, viz use of route discovery and LRTs for both datagrams and connection routing, and the adequacy of the ANS beyond the limits of coordinated addressing, are also presented, using treatment previously given in a refereed journal paper [Gur03].

Chapter 6 discusses advanced topics relating to the present approach, including a possible extension to the traditional socket API for a next generation "ANS-aware" applications; OS-like handling of network credentials and authentication as envisaged in Chapter 1; and three forms of unlimited large scale automatic network management that would be enabled by and usable under the present approach:

- ◇ Recursive, systolic aggregation of LSPs for scalability. The full scheme is given in an IBM patent application, as briefly described in Appendix B.3.
- ◇ Route-directed aggregation of IP addresses, using Dijkstra's algorithm in reverse to obtain an efficient renumbering scheme similar to Huffman coding.
- ◇ Efficient first-time, recipe-driven bootstrapping of Layer 3 in large networks.

CHAPTER 3

BASIC PRINCIPLES

This chapter explains the fundamental architectural constraint of NAT, in Section 3.1; its elegant solution comprising a functional inverse, VAS, in Section 3.2; and the basic construction and use of the ANS, which becomes necessary with this inversion, in Sections 3.3 through 3.5. The signalling procedures given will be revisited in terms of the prototype implementation in Chapter 4 and formalized in Chapter 5.

The treatment will show that end to end addressing becomes robustly available at application level without Layer 3 dependence, and that ANS and VAS are sufficient, even without route discovery within the ANS, for efficient and scalable handling of up to one intermediate "public Internet" realm. The latter is the same limit addressed in IPNL [*op. cit.*], but without changing the Layer 3 address format or using a protocol shim, and instead enabling operational simplifications in the participating end realms.

The ANS construction is generalized to more than one intermediate realm hop in Section 3.6. This leads to a triangular routing problem, as mentioned, for which LRTs and automatic route discovery is required. Discovery and use of discovered routes for both ANS routing and LSPs have been treated in the refereed paper included in the Appendix, and will be assumed in this chapter. Section 3.7 reviews the architectural significance of ANS and VAS, especially concerning the requirements of an IDP.

3.1 The architectural constraint of NAT

As will be now shown, NAT is key instance of the similarity of the current perspective to the mainframe generation of OSs, and the mechanism that continues to constrain the Internet to a globally coordinated address space. Overcoming this constraint is of crucial importance for a more flexible and mature longer term architecture Fig. 3.1 illustrates this problem of NAT with the example of two networks A and B, protected by respective NAT firewalls.

In a typical application scenario, host a inside network A, acting as client, needs to connect to a server host b within network B. For our example, say the two NAT gateways have the public addresses 129.34.4.33 and 216.239.39.104, respectively, and the two hosts bear addresses 9.2.215.95 and 10.1.1.10 within the respective networks.[1]

Assuming that a TCP connection is somehow established, the general principle of NAT is to multiplex connections at the NAT gateway for one or more hosts interior to the firewall into a single public address, or a small pool of such addresses, belonging to the gateway. Thus, a packet from host b belonging to the connection arrives at A's gateway g_A bearing the latter's public address 129.34.4.33, and is forwarded (broken arrow) to the client a using the client address 9.2.215.95, and on the reverse path, a packet from client a to server b bears the address of B's gateway g_B, 216.239.39.104, in order to be forwarded thereon to the private address 10.1.1.10 of host b.

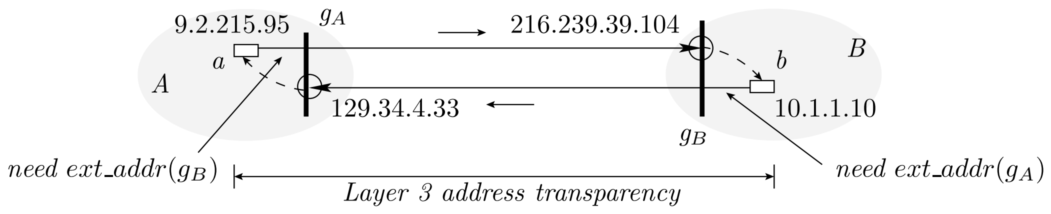

Fig. 3.1: NAT perspective

Although NAT enables the reuse of private address blocks like 10/8 (i.e. 10.x.x.x), it imposes the constraint of a single Layer 3 address space because it translates sender addresses only in outbound packets and destination addresses only on inbound traffic. Therefore, the external address of g_A, $ext\_addr(g_A)$, must remain visible between b and g_B for b's outbound data, and $ext\_addr(g_B)$ similarly visible between a and g_A for a's outbound data, so that $ext\_addr(g_A)$ cannot be reused within network B and $ext\_addr(g_B)$ cannot be reused within network A. Address transparency thus prevails at the transport level (cf. [Car00]) and limits the entire IP network, including the two end networks, to a single Layer 3 space, even if some blocks like 10/8 are reused.

1. These addresses are from actual tests from the implementation project phase of this work in the spring of 2001. 129.34.4.33 was *affine.watson.ibm.com* accessible on the Internet, 9.2.215.95 was *bubble.watson.ibm.com* inside the IBM network, and 216.239.39.104 belongs to *google.com*.

3.2 Asymmetric virtual address spaces (VAS)

A necessary first step to eliminating the single address space constraint is therefore to remove the need for visibility of public addresses in the end networks. It cannot be sufficient, as we would need an alternative means to address public destinations, but this need is fulfilled by the ANS, as will be described in the next section.

Fig. 3.2 shows how the client network address space can be isolated with *two-way NAT*, in which both sender and receiver addresses are translated in either direction. Packets are sent by the client host a with a *virtual IP address* 10.1.1.10 configured, within network A, to route to g_A, and allocated at g_A, during connection setup, to represent g_B, so that g_A knows to forward them correctly to g_B.

The outbound routing from a then no longer depends on $ext\_addr(g_B)$, so that the latter can be merrily reused within A. In fact, nothing prevents a second host c in A from bearing the same address 216.239.39.104, and still connecting to the server host b, using the same virtual address 10.1.1.10 or a different allocation, say 10.1.1.20, to once again connect to b via g_B, since the addresses have different meanings on the two sides of the gateway g_A. As depicted in the figure, once the forwarding translation states are correctly set up, there is really no need to keep the public addresses visible within A for transport. Further, the virtual address 10.1.1.10 can serve as proxy for the entire node g_B, so that even connectionless protocols like UDP and ICMP can be thus relayed, as will be described later.

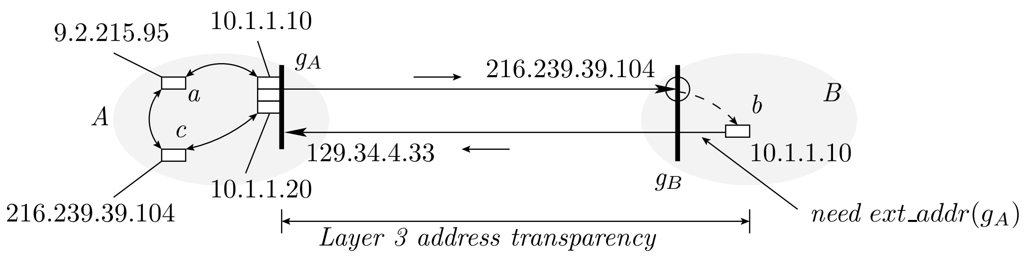

Fig. 3.2: VAS and its inherent asymmetry

Note that the above discussion mainly concerns the outbound packet routing from a to b. The inbound forwarding in NAT, as at g_B, is usually determined based on the receiving port number. This must be also the case for the inbound forwarding at g_A

because an IP-based connection is made directly from b to g_A. This inbound NAT state must hence be set up at g_A along with the virtual address, with the same NAT port number used in the outbound socket connection to g_B so that the return path packets will bear the right inbound forwarded port. This means that each outbound connection from 10.1.1.10, from a and c say, must use a different external port.

The above NAT-like use of ports is not scalable, but is needed only on the end hops that directly interface to IP routing. On intermediate hops between VAS gateways, a single TCP or UDP port number would be very likely used when if the hop is over an IP network, and a small number denoting a virtual connection or circuit (VC) id would be used for disambiguation, as will be discussed in Section 3.5. This scheme is VAS and the example shows that it is inherently *asymmetric*: the deploying side, network A in the example, gets to internally reuse public IP addresses, without regard to what is done at the remote end of connection.

This subtle change has a fundamental significance for network administration, as the deploying network can in fact reuse the entire IP address space, rather than just a few blocks, and its configuration no longer needs to be consistent with or allow for the addresses of the rest of the Internet. Further, it means that A can transparently migrate to IPv6 or dispense with IP altogether over a native MPLS cloud or a legacy ATM network, as in an AT&T ORL experiment using a modified DNS directly over ATM [PHG$^+$96]. Network B may similarly deploy VAS to avail of these benefits, but B's decision would be completely independent, with no impact on A whatsoever.

3.3 Connecting via an Addressless Networking System

The basic requirements for enabling a to *identify* the server destination b without the benefit of b's (or its gateway g_B's) Layer 3 visibility are that

$\rightarrow$ a and b have *a priori* logical connections to a third node, say e;

$\rightarrow$ b is associated, or registered, at e with an identifier unique at e, say `b`.

Client host a, or more properly applications thereon, can then identify the server host b as `e/b`, where `e` is the identifier used by a to designate node e. This is the scenario shown in Fig. 3.3 below, and still has two basic defects:

× node e must be in the public network to be accessible from behind both firewalls, but then a would be disabled from accessing e for the same reason that it cannot reach g_B via Layer 3 (note that b can still address and connect to e directly);

× e cannot direct the setting up of the virtual address (proxy) for g_B behind A's firewall g_A. (Note that no set up is required behind g_B.)

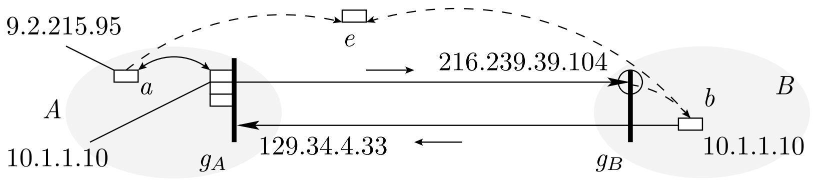

Fig. 3.3: A first step towards VAS connectivity

Part of the canonical solution, which will be proved later, is to apply the referencing notion `e/b` the other way, reasoning that by the representational principle (RP), a should be also logically visible, at least for the lifetime of the connection, to b, and should be similarly addressable in the form `e/a`. Both defects identified above can be then solved by having g_A establish a prior connection to e, as `e/gA`, say, and then having a establish a similar connection instead to g_A, which can be reached, so that a can then be referenced as $e/gA/a$. These *a priori* connections are shown as broken lines in Fig. 3.4, in which the IP addresses and node labels have been replaced by the ANS referencing scheme just described. If we installed the ANS forwarding software at each of these nodes, we would have both a global identifier space and bidirectional logical connectivity between a and b via gA and e, as traced by the broken lines.

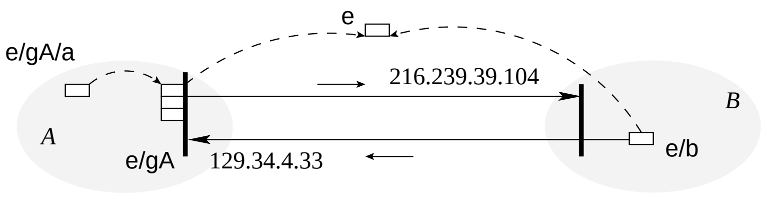

Fig. 3.4: VAS connectivity via ANS

The logical route would be triangular in going via e instead of directly between the gateways g_A and g_B, but this can be easily corrected by using the logical route to set up a more direct connection. Node e can trivially acquire knowledge of $ext\_addr(g_B)$

from any packet it receives from b, and likewise of $ext\_addr(g_A)$ from packets arriving from g_A, as if it were a Layer 3 bridge.

Should either of a and b request connection establishment to the other, the request would traverse through e using the logical route; on detecting the request, node e can simply return $ext\_addr(g_B)$ to g_A or $ext\_addr(g_A)$ to g_B, respectively, to enable the requesting gateway to establish a direct one-hop connection to the other.

3.4 Complete setup 1: single private realm

The above ideas can now be put together to completely work out the example scenario of the preceding figures. Three connections would be of interest:

- a to b, illustrating how b would be specified by the client application (on a) and how the connection set up occurs;
- a to c, illustrating reuse of $ext\_addr(g_B)$ without confusion or conflict; and
- c to b, particularly illustrating the complete separation between logical (*control plane*) and transport (*data plane*) addresses.

The control and data plane terminology relates to MPLS, where IP is currently used for the control plane and LSPs form the data plane, so that MPLS is sometimes referred to as Layer 2.5 in the OSI reference model (Fig. 1.3). In the present approach, IP is used strictly as a glorified Layer 2 data plane, with the forwarding states of LSPs above them in reverse to the current usage, as will become clear.

To illustrate the generality of the approach, only traditional IP applications will be considered, so that the difference from traditional NAT and IP will be only in the operation of the network. This means that the client application will perform a DNS lookup to obtain an IP address and use that to open a socket connection.

3.4.1 $a \rightarrow b$ – connecting to an external server

We would want the IP client application on host a to use the absolute name reference `/e/b` to specify host b, but without changing the application code. The trick, based on AVES [NZS01], is to pass to the client the name reference, in a suitably modified form, as the DNS destination name, and to intercept the subsequent DNS lookup from the client application to perform the ANS signalling.

This is just what is accomplished in the following sequence of steps with reference to Figs. 3.4 and 3.3:

B-1. Server host b connects to e, using TCP and registering as `/e/b`. This suffices to set up a NAT state in g_B and provides e with $ext\_addr(g_B)$. For this example, we will assume that either b and g_B are the same or b additionally arranges for suitable Layer 4 forwarding at g_B between b and connecting clients. This too can be handled using the ANS, as will be described in Section 3.5.

We will require b to also provide the listening protocol and port number, say as a combined text string p, with the registration.

B-2. Client host a is configured to use g_A as its default DNS server. Client application, say `clapp` that takes a command line argument specifying the destination DNS name, is then invoked on a as "`clapp b.e.`". The DNS query packet arrives at g_A, where the ANS translates it back to `/e/b`.

B-3. The ANS on g_A *interprets* the name string `/e/b` as an ANS logical path, like a directory pathname in a Unix filesystem, and forwards it to node e.

B-4. At node e, the next label `b` is looked up to recall the server registration data from the first step, comprising the tuple $\langle ext_addr(g_B)\,, p\rangle$. Node e simply returns this to g_A with a success code.

B-5. Node g_A allocates a virtual IP address, 10.1.1.10 say, from a reserved address pool configured to route, within A, to g_A, and sets up a two-way forwarding state, to forward packets arriving from the interior network to address 10.1.1.10 to the exterior network to address $ext\_addr(g_B) \equiv$ 216.239.39.104.

B-6. Node g_A then returns the allocated virtual address, 10.1.1.10, as a DNS response packet to a. When the client subsequently opens a client socket connection to the returned address 10.1.1.10, the packet addresses get translated and forward to and back from g_B. This can be arranged in either of two ways:

Layer 4 relay: g_A sets up a protocol-specific listening socket (i.e. TCP or UDP, etc. as specified in p), bound to 10.1.1.10 and relays the connections to g_B, much like port forwarding in `ssh` for TCP. The port numbers on g_A for packets bound to a are then those of the relayed connections.

Layer 3 relay: g_A listens using a raw socket and relays all packets arriving for 10.1.1.10, including TCP SYN, etc. to g_B, so that any TCP connections will literally extend to g_B. Note that in this case, the same virtual address cannot be used for a second connection to b from a different host in A, for there would be no way to separate the packets received from b.

The Layer 3 relay allows simpler forwarding but would be less secure and more vulnerable, on a Unix-like relay host, as it would require root user privileges for the raw sockets.

A Layer 4 relay would not need special privileges for most TCP and UDP ports, and could be configured to relay only specific protocols or to authenticate users. This was accordingly the method chosen in the prototype.

3.4.2 $c \to b$ – client reusing server address

It is trivial to verify that the above procedure will work regardless of the client's host address in A. This includes the value $ext\_addr(g_B) \equiv 216.239.39.104$, meaning that clients on host c will be able to connect to server b exactly like those on host a. It is useful to state the conditions and the result formally, as follows.

Lemma 1 (Consistency of address reuse). *A necessary and sufficient condition on address assignments within a realm for connecting to a server outside the realm by the procedure of Section 3.4.1 is that the assignments be consistent within the realm.*

Proof. The client host a sends or receives only in steps B-2 (sending of DNS query) and B-6 (receiving of DNS response packet and opening socket to server). The query packet must arrive at g_A, so that g_A must have a unique and routable address within A and a must be correctly configured with this address for its DNS server in B-2. In B-6, the client must receive the DNS response packet, so a's address, also configured in B-2, must be conflict-free and routable within A. Finally, when the client opens a socket to the server in B-6, the destination address given is the virtual IP address allocated by g_A, which therefore needs to be unique and routable to g_A in A. This first part proves the necessity, and holds for every potential client, including c.

The argument for sufficiency is straightforward: if any of the address assignments for a and g_A and the virtual address allocated in step B-5 recited in the first part were not unique or not correctly routable, it would break the procedure. $\square$

Client c can reuse b's public address $ext\_addr(g_B) \equiv 216.239.39.104$, and not lose its ($c$'s) own ability to connect to b, thanks to the procedure of Section 3.4.1.

3.4.3 $a \rightarrow c$ – connecting to local servers

One would expect that a fraction of the procedure should be sufficient for connecting to servers in the same realm, and should not need to involve virtual addressing. This is illustrated below for the case of client a connecting to an application on host c:

C-1. Server host c connects to g_A, using TCP and registering as `gA/c`. Its address within A, $addr_A(c)$, is retrieved from the received packet header. The protocol and port number specification, say q, is again required with the registration.

C-2. As in B-2, client host a is configured to use g_A as its default DNS server. The client application is invoked on a as "`clapp c.g_A`". The DNS query arrives at g_A, where the ANS once again translates it back to the form `gA/c`.

C-3. When the ANS on g_A *interprets* the name string `gA/c`, it identifies the top node reference as its own, so the request is not to be routed any further.

C-4. Node g_A recalls the registration data from the first step, the tuple $\langle addr_A(c)\,, q\rangle$.

C-5. As address $addr_A(c)$ belongs in A, g_A determines that it can be directly used and no virtual address allocation is necessary.

C-6. Node g_A returns $addr_A(c)$ as a DNS response packet to a. The client host a then opens a socket connection directly to $addr_A(c)$. Note that in this case, the registered protocol and port information, q, is left unused.

Since this procedure merely binds a name to an address (C-1), looks up the name to fetch the bound address (C-2..C-6) and connects to the looked up address (C-6), it is functionally identical to DNS within realm A, suggesting the following corollary.

Corollary 1.1 (Local reduction to DNS). *The procedure of Section 3.4.1 reduces to DNS functionality within the same realm.*

There are at least the following basic differences from the DNS, however:

D-1. DNS bindings tend to be static; dynamic DNS (DDNS) exists, but its use is generally limited to clients using DHCP. Secondly, IP addresses and therefore their DNS bindings invariably refer to network interfaces, not to hosts or the applications thereon. As shown in B-1, the ANS is *designed* to include sufficient application information, such as protocol and port number, for relay setup.

The DNS does not possess record types for such use. Although we could define new extensions or reuse existing record types, we would also have to introduce a *svrent_t* structure, extending the usual *hostent_t*, and rewrite client resolver libraries to pass this to clients. We would then have to rewrite the applications themselves to use this structure. The exercise would be more or less useless for networking within the immediate IP realm and only useful to gateways or ANS nodes like g_A and e on the multi-realm scale.

D-2. Even if all of the DNS servers, resolvers and applications were modified as above, the DNS was as such designed as a *database* mainly of IP addresses keyed by name, rather than as a logical routing network. It works well for relatively static use, but does not scale to rapidly changing situations: leaf-level bindings can be updated everytime, as done for load-balancing for instance but it's hard to use the DNS like a hand-off mechanism because the caching of NS (nameserver) records is crucial for its (DNS's) operability in the current Internet [JSBM01].

As will be explained with the prototype implementation, the routing approach of the ANS enables subsecond response guarantees for hand-off-like mobility – in particular, it supports SS7/GSM-like short message service (SMS) capability that can deliver successive messages to changing locations every time. This is not just a feature, but a necessary aspect of its canonical property: conversely, the DNS provides less functionality because of non canonical assumptions, and is a relatively inefficient design representing overconfiguration since DNS entries need to be configured in addition to the server IP addresses.

D-3. As a distributed *name* $\mapsto$ *address* mapping database without routing semantics, the DNS does not associate name bindings with network locations. There is no analogue of registering a name binding on a specific ANS server. Instead, the DNS tends to copy name binding data from the authoritative nameservers of zones where the names are configured in, to other name servers as cache data, in order to provide the performance we have today. Caching does occur in the ANS indirectly in the form of LRTs entries, not as replicas of the binding data.

Correspondingly, DNS servers tends to hold large numbers of cached bindings from all over the Internet, whereas an ANS server holds only the name bindings bound directly on it, the name registrations of its immediate child nodes, and its LRT entries, which would be typically kept in separate hashes for fast localized search directed by the parsing of the destination address string, i.e. the local bindings would be looked up for a name label only if the destination address prefix matches its (the server's) own address.

This lighter, faster approach is appropriate to its routing role, or in other words, an ANS server functions and is designed more like a router than a database.

D-4. The mapping database design also explains why *reverse-lookup* is unreliable in the DNS, and imposes more configurational effort: the reverse mappings must be separately configured, and can be inconsistent with the forward mappings. It also tends to be slow because the reverse mappings would not be automatically cached by the forward lookups.

In the present approach, the sending node's ANS name must arrive with each datagrams, making a reverse lookup unnecessary. If an LSP is set up, as in the current examples, the remote ANS address could be easily retained at the VAS gateway for the duration of the connection for efficient retrieval.

D-5. Lastly, the DNS was designed for a single flat IP realm and does not elegantly extend to multiple realms like the ANS. Extensions of the DNS for multiple realm use have been proposed in both IPNL [FG01] and TRIAD [CG00], neither of which uses gateways or another means to construct independent absolute identifiers. Both schemes entail more configuration effort, requiring configuring of the realm IP address spaces, the DNS within each realm and additional inter-realm zone data to extend the DNS visibility across realms. IPNL simplifies the scenario by restricting itself to a circle of edge realms around a central realm, but calls for extending the Layer 3 address format to reach any foreign destinations. The client protocol stack, resolver library and application software must all be extended, but the result would still be a finite, albeit larger (48-bit, as opposed to 128-bit in IPv6), effective Layer 3 space.

The ANS promises simplifications in the IPv4 Internet itself, i.e. even without its use for multi-realm networking in the sense of IPNL and TRIAD.

3.5 Complete setup 2: between private realms

Fig. 3.5 illustrates connections between IP hosts in separate private realms, extending upon the scenario of Fig. 3.2 primarily by replicating the realm configuration of A in B. Within either realm, the peer host needs to be represented by a virtual IP address routing to the local gateway; we can assume that the same numeric value, 10.1.1.10, happens to get used at either end as the virtual address for the other, as shown.

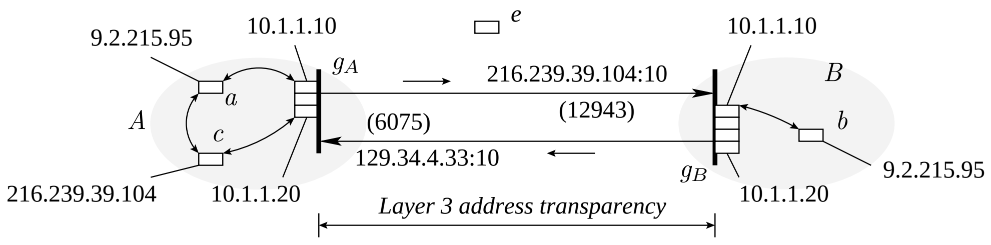

Fig. 3.5: VAS connectivity across realms

The virtual addresses are adequate only for routing outbound packets, however. For inbound packets, we no longer need to dedicate a port for every connection, as in

NAT, since the incoming connection is established by the peer gateway itself rather than by the peer IP application host. Moreover, the number of hosts in each realm is itself likely to grow considerably from current levels, and the present approach would encourage such growth by permitting full reuse of the IPv4 space (or of IPv6 for that matter). Even more significantly, when the approach is deployed over large networks of realms, as envisaged for the long term, each gateway would be relaying not only for hosts within its immediate realm, but also connections passing through, unlike NAT.

We in fact need *the same kind of generalization of NAT for IP that MPLS made for tunnelling over ATM*: ATM tunnelling and NAT have been *opportunistic* uses of features that already existed, viz the AAL5 VPI/VCI provision and the port scheme in TCP, respectively. We need generalized tunnels that do not depend on such specifics, which means using LSPs *above* IP, instead of the other way around in the current use of IP as the application support and control plane for MPLS. This is illustrated in Fig. 3.5, where port 10 (currently unassigned in TCP) is used for the VAS tunnels, and the VC numbers identifying the tunnels are shown in parentheses.

The change in the procedure of Section 3.4.1 is then straightforward, as below.

E-1. Gateway g_B connects to e and registers as `/e/gB`, instead of the registration of b on e in Section 3.4.1, supplying its protocol and port data, r. Correspondingly, node e records the tuple $\langle ext\_addr(g_B)\,, r\rangle$.

E-2. Server host b cannot register on e directly because it cannot connect to e using IP (or an infrastructure network method), as $ext\_addr(e)$ would be likely reused within B and unroutable outside of B. However, we can have it connect to g_B and registers thereon as `gB/b`, along with its listening information p. Like e, g_B records the tuple $\langle addr_B(b)\,, p\rangle$, and the full path for b then becomes `/e/gB/b`.

E-3. DNS configuration on client host a – same as B-2. For starting the client on a, we must now use "`clapp b.gB.e.`", reflecting the new full path.

E-4. The path gets inverted back to `/e/gB/b`. Routing of the DNS query from a is same as in B-3 up to node e. At node e, the query gets routed further along the downward link to g_B, which is identified by the registered information at e as the next hop on the logical route for the given path name.

E-5. At g_B, the onward label `b` is looked up to retrieve the server registration tuple $\langle addr_B(b)\,, p\rangle$. As b is the final node identified by the path name in E-3, there are no onward hops. However, g_B cannot return $addr_B(b)$ to e because $addr_B(b)$ belongs inside realm B. By the VAS principle (Section 3.2), a foreign peer must be represented by a virtual address in B and a VC on the intermediate hops.

Accordingly, g_B allocates a virtual IP address, 10.1.1.10 say, very similarly to step B-5, and a VC, 12943 say, sets up forwarding between the two, and returns the $ext\_addr(g_B)$ to e. As $ext\_addr(g_B)$ is in the same realm as $ext\_addr(g_A)$, it is relayed by e to g_A.

E-6. Gateway g_A allocates a corresponding VC, 6075 say, and a virtual IP address, 10.1.1.10, as before for the interior routing, and sets up forwarding between these. It then sends a handshake packet bearing the allocated VC 6075 using the received address $ext\_addr(g_B)$, thereby completing the two-way linkage of VCs to create an LSP between the gateways.

E-7. Node g_A then returns the allocated virtual address, 10.1.1.10, as a DNS response packet to a, as before, completing path set up. Host a may then open a socket as before to the returned virtual address 10.1.1.10 and get connected, via the forwarding states at g_A and g_B set up above, to server host b.

Note that unlike the case with NAT, the LSP between g_A and g_B will relay *any* kind of packets between a and b, including UDP, ICMP, or raw IP. The VAS channel is a bit pipe between the virtual IP addresses in the two end realms.

The behaviour of this pipe is dictated by the base protocol between the gateways: if TCP, as suggested above, the pipe would be reliable and order-preserving. If UDP were used, not only would both reliability and order be compromised, but the integrity of point to point communication would also be lost, as any public node would be able to inject packets into g_A.

Using the new protocol SCTP [SMS$^+$00, OY02], we could preserve the integrity, but trading off reliability and order for timely delivery, as required for media streams. It would also allow multiple paths between the gateways, say for different QoS needs.

3.6 Connecting across many realms

Three difficulties have had to be solved to make it useful to extrapolate the procedure of Section 3.5 across multiple intermediate realms, the first two being:

- We would need an intermediate node like e in every such realm, which means finding such a participating node within each realm. An efficient choice of such a node would be difficult to make over any set of substantially complex realms. Moreover, the path string given an arbitrary choice of intermediate realm nodes would not be an absolute address: we could get paths like `a/A/B/C/.../b` which would be no better than a reincarnation of UUCP. This appears to be just where TRIAD has devolved into DNS based routing research (see [GC01]).

  This problem is itself another symptom of the bottom-up perspective, since it envisages choosing the sequence of intermediate realms first and then seeking forwarding nodes within each. The more basic question that should have been asked is how the user (or application) could at all identify a specific destination in the first place, as *UUCP-style paths are not absolute references.*

- A second problem that emerges in the multihop context is triangular routing. In Fig. 3.6 below, for instance, an available direct path, say a SONET connection between hosts a and b bypassing several realm boundaries, would get overlooked by the automated procedure following the ANS path (finely dotted).

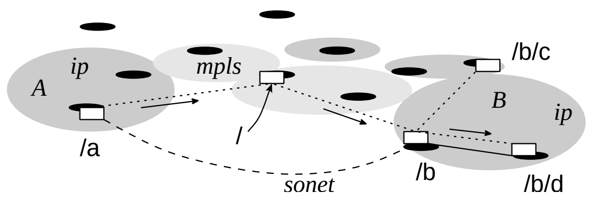

Fig. 3.6: Multirealm example

The present solution is that the intermediate realms would be chosen by traversing an ANS path, directed by an absolute destination path, rather than by specifying the intermediate realms first. Fig. 3.6 shows such a scenario, in which client on host *a* in one IP realm *A* is given the application path `/b/d` to connect to a destination host in a second IP realm *B* by following ANS links (dotted lines, arrows) first to the root node and thence to node `/b`, after traversing non IP links and an MPLS realm, which may or may not be running IP at all, in the process. An intermediate realm need not even have a node registered in the ANS, and would then not be identified in the ANS paths, unless it has nodes needing to participate in the ANS network as destinations, since pass-through links can be set up across its boundary gateways. Also, all realms are equal in the approach except as ordered by their positions in the ANS hierarchy. The ANS location can be changed by disconnecting and re-registering at a different position. In general, nodes, and their realms, connecting at higher levels would likely need to deal with greater traffic, but even this would be far less than the root server load in the DNS because of the localization inherent in the hierarchical routing. The traffic would be further redistributed by non-ANS routing, discussed next.

The only way to solve the second problem is by independent route discovery, just as in IP. The discovery process would populate LRTs at nodes *a* and *b* with entries of the form `/b/d` $\mapsto$ *sonet_link* and `/a` $\mapsto$ *sonet_link*, respectively, where the token *sonet_link* denotes applicable parameters. Then, at the starting node (*a*) as well as at each intermediate node (e.g. the root), the LRT must be looked up to see if a more direct route is known, by looking for the longest prefix match. This is the same as in IP routing, except that the addresses are path strings and not numeric.

There are differences from IP routing, however. Firstly, in the present Internet, the inter-domain topology is mostly flat and there is little logical hierarchy to use as default; this makes discovery critical for operation of the Internet. NIRA proposes to change this situation to ensure hierarchical default routes, similar to that advocated here, and the present thesis will hopefully make this case more compelling.

Secondly, the default bANS routes would be representative of provider routing, and hence efficient anyway, the same reasons as in NIRA. One would not use the base ANS directly in applications, but the dANS references would map, through LRTs, to the bANS routes. The availability of fairly efficient default routes poses opportunities as well as more questions that need to be studied. An example is the opportunity for "load shedding" packets, after only a cursory examination of their addresses, to some other ANS node that appears unlikely to pass the packets back, with the confidence that the latter would be able to route them appropriately, thus potentially creating a "social services problem" in its vicinity.

Thirdly, there is a possibility of discovering direct links, like the SONET link in the previous figure, on the fly, even as a connection request is being routed through the default ANS route. The idea is to determine, along the request's path, if an IP realm boundary gets crossed. If not, then the destination is in the same realm so its IP address can be used directly, as in Section 3.4.3, avoiding a separate discovery. Further, if an even number of realm boundary crossings is detected, a check could be made to see if there was a return to the original realm. This would call for configuring a realm id number, which would put a theoretical bound on the size of the reachable Layer 3 address space, but would still permit a very large number of realms. Clearly, no analogue of these ideas is possible with IP alone.

The third difficulty initially faced, as mentioned, concerned the visibility of the provider chain in the base ANS names, before the solution of derivative links became clear. The problem was that this would make the ANS *worse* than IP addressing, by not only reintroducing multihoming and mobility problems, but by doing so at the naming level, which would lead to users and applications depending on the provider chains of remote servers in a big way. This problem has since been solved, at least in principle, by the scheme of derivative links and subscribed updates. This scheme appears to be a more elegant and scalable solution to the general problems of mobility and multihoming than prior proposals including FARA [CBFP03], in that the latter lack any similar capability for automatic change subscriptions.

3.7 Architectural significance

The three basic components described above, VAS, LSPs and ANS, are significant, as discussed below, even without lateral route tables (LRTs), route discovery, derivative links and the filesystem-like control and administration aspects to be discussed later.

The ANS suffices both as a highly dynamic and scalable, yet lightweight messaging name space, and as a versatile "universal control plane" that can be very easily set up to operate *securely* and *bidirectionally* across any number of firewalls or network boundaries, including non-IP networks like ATM or MPLS, as shown. This is more general and substantial in scope, yielding greater homogeneity and simplicity, than existing signalling protocols such as SIP [HSSR99] and their requirements as currently envisaged in the IETF [(Ed04], which are both largely concerned with the traditional difficulties of firewall opacity.

The homogeneity and simplicity of the ANS in this role are due to the fact that it handles inter-domain addressing at the level that directly concerns the reason for the administrative domain boundaries, viz ownership and authentication, whereas in the current Internet architecture, both these basic functions need to be obtained by negotiation from a lower layer (IP), which was designed without such boundaries in mind, like older host OSs that did not support user authentication and access control. NAT, and IP's fixed-length form that led to exhaustion concerns and motivated NAT, have been generally blamed for the loss of Layer 3 transparency and the hurdles as well as routing inefficiencies resulting from the NAT firewalls. However, the security of firewalls is a basic need for any serious use of the networks.

The ANS operates at precisely the level at which authentication and access control can be best applied and administered and identifiers useful to applications and users are present, so that the Layer 3 address boundaries vanish in significance. This is as desired for an IDP, as described in Section 1.1, and the inappropriateness of Layer 3 in this role as currently prescribed will especially clear from the discussion of the actual demonstration scenarios in Section 4.2.

It has been shown, however, that the ANS symbol lookup can be offloaded to a DNS within the same realm. This is useful primarily when setting up ANS links, as will become clear from the discussion of ANS node initialization in Section 4.1; most other uses of DNS queries above have been for supporting unmodified IP applications by overloading the usual DNS lookup with ANS routing and path setup. This is not the ideal interface, and future applications would want alternative protocol interfaces or APIs more attuned to the overall approach. Two such APIs have been attempted, viz a system calls API implemented on AIX [Gur00], and a Unix virtual file system (VFS) implementation, the INFS, as mentioned.

A generalization of DNS lookups for the routing itself, i.e. not simply for the ANS links, is considered in Section 5.6, and permits a closer comparison, in Section 5.7, with both TRIAD and IPNL.

CHAPTER 4

IMPLEMENTATION

A prototype of the basic architecture above, comprising the ANS, signalling for LSP setup and VAS in the form of an application level gateway (ALG), was implemented in C++ as the object of an initial project undertaken in Spring 2001.

The first result of this project was addressless short message service (SMS) (without LSP and VAS) demonstrated at NYNET'2001 [Gur01a]. Subsequently, addressless TCP (`talk`) and ICMP (`ping`) were demonstrated in June 2001 at the Internet Society Annual Conference, INET'2001, in Stockholm [Gur01b].

The full implementation is included in the Appendix. The implementation and its detailed operation are described below, illustrating the signalling procedures of Chapter 3 and more particularly the fundamentally high level addressing and routing as required for an IDP, per Section 1.1, and for OS-like usability and automation, as envisaged in Section 1.5.

4.1 Description of the prototype

The prototype implementation made use of an older "generic server" class library for HTTP-like protocols, including HTTP itself[FGM$^+$99] and the RTSP [SRL98], which provided working class definitions for `IpServer`, `UdpServer` and `TcpServer`, had several nice features of the Apache web server, such as pre-forking, built into it, as well as a simple, easy-to-verify shared memory allocator for safer, more general data sharing than by threads.

The design included a base RTSP server, from a prior involvement in a reference model implementation of this protocol, that concurrently supported multiple backend applications by dynamic loading of library modules via a configuration file. Any of these applications could crash, be terminated or reloaded independently, as each was run with a separate shared memory segment. The experimental backends included

- an interface to IBM's VideoCharger product executing on another host, written by a colleague as a test of inter-operability, and
- a lightweight multi-player, multi-engine "game server" that provided limited online access to Deep Blue Junior chess engines for several years.

It made sense to reuse this server for the present work as the code was designed and written solely by this author (except for the VideoCharger backend as a sanity test), was experimental to begin with, already suffered from short-term design workarounds for AIX/xlC and Linux/g++ compiler quirks of the time, and had been benchmarked for RTSP, showing that it could handle several hundred RTSP session setups and teardowns per second with no observable drop in performance.

Though UDP was supported by the RTSP server, TCP was preferred as the link protocol, both to avoid issues of reliability and to achieve bidirectional ANS signalling, i.e. inbound flow set up requests, across a corporate firewall without bending the usual firewall rules, to demonstrate inbound addressing and connectivity.

RTSP was chosen for setting up the ANS links, largely because, despite its name, it is not fundamentally specific to streaming multimedia data. Rather, it is a mature, convenient protocol for negotiating and managing sessions for a much larger class of applications, as had already been demonstrated by the game server backend.

The main difference between RTSP and the SIP [HSSR99], pertinent in the present context, is that RTSP is a "one-hop" protocol for connecting IP end points directly, whereas SIP uses email-like addresses and is meant to be routed over firewalls on its own. SIP thus competes with the ANS with regard to inter-domain routing, and was therefore useless as an a link setup protocol when demonstrating ANS. The sessionizing made it possible to bring up the links in any order, and to reestablish them automatically after a preset or programmed interval if they broke. The RTSP protocol methods employed are listed in Table 4.1.

Another reason for using RTSP was that it enabled a simple form of authentication for security. Although a transport layer security (TLS) scheme, such as secure sockets layer (SSL), was not implemented in the RTSP code, some security could be achieved with RTSP by simply storing the corresponding session description files, per SDP [HJ98] specification, on a separate Apache web server under password protection.

Method	*Action*
OPTIONS	Return the list of methods
DESCRIBE	Return SDP[HJ98]
SETUP	Create a session object
TEARDOWN	Destroy session if present

Table 4.1: RTSP control for ANS links

The sample configuration file listed as Fig. 4.1 illustrates these and other practical considerations such as various timeouts. The parent address must be within the same domain, to be reachable without inter-domain negotiation, and hence may be specified by its name in the local DNS. The configuration included, as shown, two additional TCP ports for child ANS registrations besides the RTSP port for listening to child ANS session requests, and the basic server port for "vanilla" ANS child registrations:

- "switches" providing the LSPs, comprising ANS nodes with the appropriate setup/teardown signalling capabilities;
- and host interfaces, comprising switch nodes with additional functions to provide VAS terminations and to intercept and respond to (client side) DNS queries as described in Sections 3.4, 3.5 and 3.6.

LRTs and route discovery protocols would be implemented in all three classes. This separation is illustrative, but was dictated by the slightly different requirements for both SDP and authentication.

The ANS link protocol was based on HTTP and RTSP as well for ease of debugging and illustration. Each protocol message accordingly comprises a block of ASCII text lines ending in a blank line, with a command or response header as the first line. The example in Table 4.2 below is from an actual demonstration log file.

In the example, the first "URL" `/B/A/p/p1` is the ANS destination address, `CSeq` is the sequence number of this interaction like in HTTP and RTSP, and last header in the response (`Medium`) contains four parameters, ending in `address`, in one line. The address returned in the response, `127.1.0.1`, indicates that the `127/8` address pool reserved for loopback was used for VAS, consistent with the mapping scheme of Fig. 1.6 (Section 1.5, on page 19).

```
# @(#)Rtspd conf for acsd              # ACS = old name for ANS
                                       #   (addressless connection system)
Basedir         /home/prasad/prj/osix   # Working directory
Port            8554                   # RTSP port to use, should come first
#Uid            :invalid:              # to run as
#Gid            :invalid:
Timeout         60                     # max backend response; sec
Preforks        0                      # of RTSP listeners
Logname         log/rtsp.log           # local to working directory
Errname         log/rtsp.err
Library         lib/ix86/libtools.so
Backend acs/* lib/ix86/libacsd.so {     # dynamic library of backend module
        Location        me             # this node's label
        Address         9.2.215.13     # parent to connect to
        SweepEvery      60000          # in ms; for unconnected sessions
        NoTimeouts                     # for connected but inactive sessions
        #AllowAny                      # only RTSP SETUP requestor may connect
        IpcPath         etc/acsd.cf    # shared memory for session data
        IpcMode         0600
        ShmSize         16384          # 16 MB
        LocalTime                      # timestamps in logs
        Resetlogs       true           # else append
        IoLog           log/acsd.io    # filenames
        DebugLog        log/acsd.dbg
        ServerPort      8085           # for general ANS node to connect
        HostPort        8086           # listen for hosts only (leaves)
        SwitchPort      8087           # listen for switches only (leaves)
        IdleTimeout     300            # in s; drop TCP if unused this much
        BusyTimeout     180            # in s; ANS protocol timeout
        HostTimeout     60             # in s; specially for host links
        SwitchTimeout   10             # in s; for switch links
        ServerTimeout   5              #
        MaxServers      10
        HostKeepalive   0              # in s; send a nul every so often...
        SwitchKeepalive 0
        ServerKeepalive 0
        Map server/*    file://home/prasad/prj/osix/etc/rtsp/* # URL->SDP
}
```

Fig. 4.1: Sample ANS node configuration

Basic tests of the protocol involved binding and retrieving simple data objects, which would lead to the notion of using the ANS as a distributed resource filesystem, described in Section 1.5. This operational primitive is invoked by a command header, `BindMsg:`***`addr`*** with payload data given as a `Message:`***`text`*** subheader following the `Cseq:` and `From:` subheaders. The command causes the packet to be routed to the node specified by the directory part of the address, e.g. `/B/A/p` for address `/B/A/p/x`, and stored in memory under the basename part of the address, "x". An `Ack:`***`retaddr`*** or `Nack:`***`retaddr`*** response is returned, routed by the return address given in the `From:` header of the request. Subsequently, a `Get:/B/A/p/x` command issued from

Request	*Response*
`Get: /B/A/p/p1`	`Ack: /B/q/q1`
`Cseq: 11`	`Cseq: 11`
`Hops: 2`	`Hops: 4`
`From: /B/q/q1`	`Medium: direction=duplex;`
blank line	`linkage=client;`
	`protocol=tcp;`
	`address=127.1.0.1:9911`
	blank line

Table 4.2: Sample ANS exchange for VAS

any ANS node retrieves the stored message, again as a `Message:`***`text`*** subheader.

The other commands implemented are `Bind`, for creating the service registrations as described in Sections 3.4, 3.5 and 3.6, and primitives for creating and destroying forwarding states, used in the demonstration scenarios to be described in Section 4.2. These primitive operations have been rigorously treated in the refereed papers [Gur00, Gur03] included in the Appendix. The primitives are dynamically loaded in the switch and host instantiations of the ANS server, from slightly different libraries, and are simply relayed by the other ANS instantiations in the network.

The ANS registration itself involved the child node first connecting to the parent using the protocol, address and port specified by the `Transport:` header in the RTSP response to a session `SETUP` request. The connection logic only makes sense with a connected protocol like TCP, so the SDP files for the RTSP URIs invariably specified TCP. The child was required to output a one line `Session` ***`id child`*** message to the parent immediately on getting the socket connection, where the session id would have been assigned by the parent's RTSP server and included in the RTSP response, and *child* was the name specified in the Location entry of the child's configuration (line 11 in Fig. 4.1). If the session id matched a still unconnected open session *and* the child name was not already in use, the parent would respond back with a `Location:`***`addr`***, telling the child its own ANS address, and mark the session as connected. Otherwise, the connection would be silently dropped by the server.

All states had timeout settings for security: unconnected sessions, created by the RTSP SETUP or because a child unexpectedly got disconnected, would be destroyed if they remained in the unconnected state for too long (typically 5 seconds), and even connected sessions could be timed out for inactivity or after a maximum elapsed time. This was later extended to the VAS address allocations, along with a “cool off” period setting to make inadvertent misrouting improbable. A very similar logic is used in the management of System V inter-process communication (IPC) ids [Ste90], which were incorporated in a system calls API in an earlier related experiment [Gur00].

The ANS nodes were programmed to retry connecting to a parent endlessly, unless a node’s configuration indicated that it was to be root (by omitting the parent address, in line 12 of Fig. 4.1), so as to yield a fairly persistent “global TCP grid”.

4.2 Example scenarios and objective

The example configurations used to demonstrate ANS and VAS typically comprised:

- ◇ legacy IP client and server applications using unmodified OS protocol stacks;
- ◇ LSP-LSP ANS “switch” nodes;
- ◇ a server-side VAS-capable ANS node; and
- ◇ a DNS-intercepting client-side VAS-capable ANS node;

The object was to demonstrate bidirectional protocol-agnostic addressing and routing across an infrastructure (Layer 3) addressing domain boundary, thus without using infrastructure support for addressing. For this, an independently managed corporate firewall was used with permission as the addressing domain boundary, since it blocks all incoming TCP connection requests, and does not forward UDP or ICMP. Adequate security was assured since the ANS forwards only to actively registered nodes, only Layer 4 relaying was implemented, the registrations were kept active only for the brief periods of the demonstrations, and all packets were monitored and logged.

The applications chosen were `tcptalk`, a simplified form of *telnet* that can run as server or client, for demonstrating the handling of connection-oriented protocols, and `ping`, to show the operation of a “port-less” and connectionless protocol, which could be just as well applied to raw IP with Layer 3 relaying.

The two basic demonstration scenarios used are shown in Figs. 4.2 and 4.3. Fig. 4.2 illustrates basic *subscribed inbound addressing* across a Layer 3 address boundary or firewall, not for “puncturing a hole in the firewall”, which can be done rather easily by relaying IP over HTTP [GB01] say, but as a simpler, more general, and more scalable signalling mechanism than SIP, that would be correspondingly easier to configure and secure, and to demonstrate its independence from Layer 3 addressing.

Specifically, in current technology, inbound (application) datagrams or connection requests, can only arrive in the form of UDP or TCP SYN packets bearing Layer 3 addresses. Assuming the addresses belong to interior servers and the packets somehow get routed correctly all the way to the firewall router, the latter would generally not be able to validate their payload and match them to the interior server protocol, port and authentication requirements if any. Currently, therefore, server applications must have their UDP or TCP listening interfaces outside of the firewall or expressly configured into the firewall forwarding tables, and are invariably split to allow their database backends to remain protected behind the firewalls. Sun’s J2EE technology, for instance, is designed around this split.

More complex inter-business services require Virtual Private Networks (VPNs), and are otherwise limited to Web services like Webex(`http://www.webex.com`), where the data to be shared is typically hosted securely at a neutral web site.

The present approach enables application level addressing for both datagrams and connection requests, independently of Layer 3 access, and as a well defined protocol independent of a specific application. Connection requests will arrive, as explained by the example procedures of Chapter 3 and will be shortly illustrated, with application level addresses of the interior servers as an application level protocol payload.

In the most basic scenario of Fig. 4.2, a server-side VAS-enabled ANS node A, is an application level gateway (ALG) for the private network on the left of the firewall. It makes a TCP connection to an LSP-switch ANS node B in the public network, to establish an ANS link, becoming B’s child in the ANS topology and acquiring ANS address $/B/A$, assuming B is root.

Thereafter, connection requests from public network clients would arrive over the TCP link as ANS requests, as described for the inbound arrival at gateway g_B in Section 3.5 (see step E-5). There would be no need to split the server application as in J2EE, or to open a “hole” in the firewall.

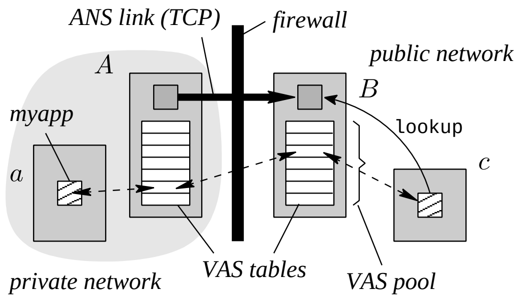

Fig. 4.2: Inbound addressing

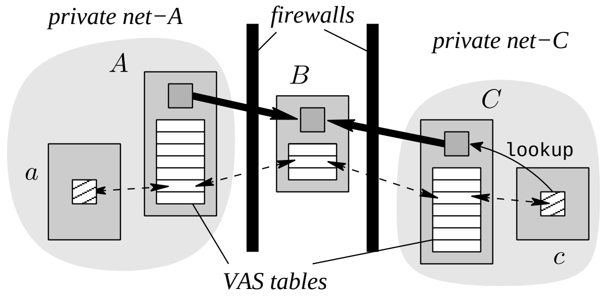

Fig. 4.3: Interdomain addressing

Furthermore, each server application gets to decide, at host process and message level granularity, whether, when and for how long it wants to be exposed to incoming requests, by registering or unregistering its service name binding, termed "listening information" in Chapter 3. The server application also gets to choose *how* its name would appear, i.e. whether the name is bound on the interior node A, the public node B or elsewhere in the ANS network, since retrieved binding will route the requests onwards to the server, like the "third-party node" e in Chapter 3.

There is, however, a minor problem with this configuration, viz that as VAS is designed for the client side network, the public node B would need to reserve a VAS pool within the precious public address space in order to support legacy (IP) clients. The only way out is for clients in other private networks to run their own ANS nodes and connect to B, and for client hosts directly in the public networks to run the ANS node software themselves. In the latter case, the local ANS service would be best configured to use the 127/8 reserved loopback address block as the VAS pool, as described in Section 1.5 (Fig. 1.6, on page 19).

Both cases are equivalent, as will shortly become clear, to the inter-domain scenario shown in Fig. 4.3, illustrating *subscribed 2-way addressing* between private networks. In this, a client-side VAS ANS node C, connects to node B as a child from the second private network, acquiring the corresponding ANS address $/B/C$. Its own network is configured to use C as the default DNS nameserver, in order to intercept queries to an ANS pseudo-domain, say `.ans`. In turn, C is configured to use a *real* DNS server for all other lookups. More general splicing of DNS and ANS lookups will be treated in Chapter 5.

4.3 Basic sequence of actions – ICMP

The demonstration begins with a service name binding *myapp* being created for the application server on private host a by invoking, on a, a special OS command `cbind /B/myapp params`, where the parameters include any application-specific arguments. (The name `cbind` relates to an experimental systems calls API on AIX called Contexts [Gur00] that led to this work.)

For example, for `ping`, the invocation was just `cbind /B/pingme icmp`, and for `tcptalk`, it was `cbind /B/pingme tcp:9999`, etc. This bind request is routed, via the ANS protocol, to the node identified by the directory part of the address, node B in this case, along with the return ANS address `/B/A/a`, the return infrastructure (IP) address, and the arguments to the bind command. At node B, a service name binding is created with the arguments and the return address, and an acknowledgement is returned to the `cbind` command process on host a, again via the ANS.

As mentioned in the preceding section, the service name binding could be created at virtually any ANS node, including A (using the corresponding address `/B/A/myapp` in the `cbind` command), or a third party node like e in the examples of Chapter 3, without impacting routability from the client. The choice would depend on the desired logical association of the service with the node selected for the service name binding, as "write access" to portions of the ANS could be restricted in filesystem-like manner.

For example, it might be easier to bind on A, than on B or on a private third-party node like C of Fig. 4.3, as the latter could require authentication based on a service agreement or policy. Users would perceive a binding under C, as $/B/C/myapp$ say, as authorized for their local use, and external bindings like $/B/myapp$ or $/B/A/myapp$ to require authentication.

The next step in the demonstration consists of the client application being invoked on a client host c with the corresponding DNS-style address `myapp.A.B.ans` as its destination argument. This causes the client application to issue a DNS lookup for the name `myapp.A.B.ans`, which is intercepted by the ANS node C, that first translates back to the form $/B/A/myapp$. Node C then executes `Get:/B/A/myapp`, as described in Section 4.1, causing the request to be routed to node A, where the service name binding is retrieved using the name *myapp*.

Using the retrieved parameters, node A opens a client socket pointing or connecting to the server application's listening socket on host a. For `ping`, the parameters specify `icmp` as the protocol, hence node A opens a raw IP socket, associates a pointer to the retrieved server address data with this socket, and then returns an acknowledgement packet with an index to this socket to node B. Node B creates a forwarding state with this index, and replaces the socket index in the acknowledgement packet with an index to this state, before passing the packet onwards back to C. Node C allocates a VAS with outbound index set to the received index, creates a raw IP socket, sets it to listen on the allocated address, and returns the allocated VAS address to host a as a successful DNS lookup response. This completes the forward data path setup with the acknowledgement, which will be formally treated in Section 5.4.

When `ping` application subsequently sends an ICMP packet, this gets intercepted by C's raw IP socket and relayed by nodes C and B over to A. A control command is sent alongside to construct the return path. Node A uses the retrieved parameters to send the ICMP packet, via the raw socket previously opened, to host a, so that a's response will be delivered by IP back to the same VAS address, to be routed all the way back to C along the return path. Node C then returns the response to host c via its previously opened raw socket. The above sequence of actions is summarized by the following steps which are indicated in Fig. 4.4.

F-1. Client at host c looks up *pingme.A.B.ans*. Its DNS query is intercepted by ANS node C and transformed into an ANS `Get` request.

F-2. The request is routed to node A, where the name binding *pingme* is looked up.

F-3. An IP address from the VAS pool is allocated, a raw socket is opened and bound to this address as a listener. The socket descriptor is added to a *pollfd* array used for the `poll(2)` system call.

F-4. A *forward* channel is allocated along the way back from A to C.

F-5. At node C, an IP address from the VAS pool is allocated and internally linked to the forward channel. A raw socket is opened and bound to this address for listening. The address is returned as the DNS response to the client on host c.

F-6. Client on c opens a raw socket and sends an ICMP packet to this address, which gets relayed through the forward channel and the raw socket on A to host a. The ICMP response from a follows the return channel set up on the outbound by an inband set up command. The raw sockets and their VAS addresses are reclaimed after a timeout.

The figure also illustrates what happens to DNS queries for domains other than `.ans.` – these are silently forwarded to the *real* DNS nameserver (step 1' in the figure), with the source address set to the real client addresses, so that the responses return directly to the client hosts (step 5').

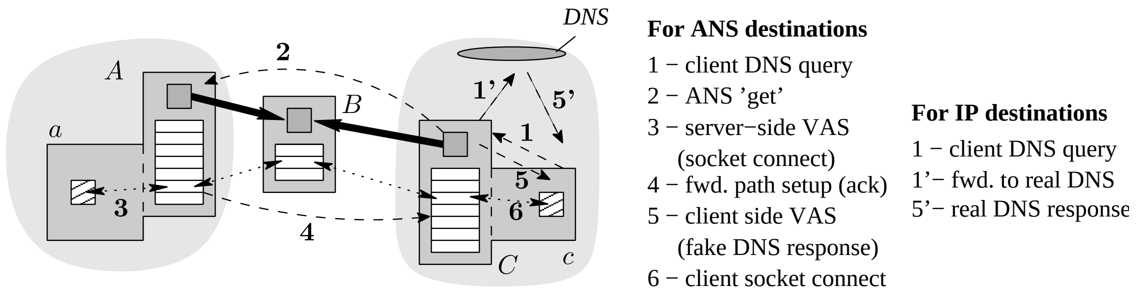

Fig. 4.4: VAS address establishment

Relaying of connectionless protocols has inherent difficulties, however, as remarked in Section 3.7 and to be revisited in Section 4.5.

4.4 VAS setup for TCP

The handling of `tcptalk` would be similar, except that the prototype uses an actual TCP socket for each VAS connection, instead of a single raw socket for all connections to a given destination. This choice was made primarily to simplify the implementation, and to avoid bugs and security issues.

The main issue is that applications like `ftp` try to open secondary connections to the same destination IP address. This is also the case with Web browsers and related software like the various Mozilla derivatives, as they include their own DNS resolvers and *cache* the destination IP addresses. Supporting this latter class without triggering lookups for already accessed addresses requires allowing secondary connections using the same VAS address. This is already supported in the prototype because in order to `accept` the first TCP connection request from the client, the prototype VAS gateway as such needs to open a TCP server socket bound to and listening on the returned VAS address, analogous to the listening socket for ICMP in step F-3 in Section 4.3.

For each new connection that this listening socket accepts, the gateway allocates a new channel all the way to the server side gateway, which correspondingly opens a new TCP client connection to the application server host. These steps are analogous to steps F-4 and F-3 respectively, in terms of the direction of the socket connections. However, the channel signalling is in-band and the client-side socket connection occurs first. In any case, the gateways at both ends maintain a list of active TCP sockets for each VAS address for "garbage collecting" the VAS pool.

4.5 VAS setup for UDP

The handling of UDP is simpler as there is no similar notion of multiple connections to be supported. Each UDP port represents a separate application context, and must be negotiated as such via the ANS, corresponding to the initial steps F-1 through F-4 of Section 4.3. The only optimization allowed, also applicable for TCP, is the use of a single VAS address for services hosted by the same server host, as identified by the respective service parameters. As mentioned in Section 4.3, the server's address is automatically stored with the service parameters, and therefore available for ANS routing all the way to the server-side gateway in step F-2.

Ordinarily, the main difficulties with relaying UDP are that the sender's address

- would be lost unless passed to the receiving host by a separate channel,
- and would be meaningless outside of its original domain.

The sender's address could be retained by including the IP header with the relayed payload as in Layer 3 forwarding defined in Section 3.4, but replies from the UDP server, routed by the original sender's address, would be unlikely to reach the server-side relay gateway since the address is inappropriate within the server's realm. There is, remarkably, a TLS scheme for UDP [MR04], but no satisfactory relaying solution!

It should seem remarkable that this difficulty does not occur with ICMP, which is an even simpler connectionless protocol. The reason is that in the `ping` application illustrated in Section 4.3, there is explicit association of the server's response packet with the last "ping" delivered, so the relay gateway can safely assume the identity of the sender. The association is enforced by destroying the LSP after a brief timeout. This works well because the Unix `ping` application correspondingly repeats the DNS query and socket connection for each ping, and the `ping` is not invoked too often.

With UDP, there would likely be packets from multiple senders arriving rapidly at the server. The only way to ensure that its responses are forwarded to the relay gateway despite the original senders' addresses is to alter the IP route table at the server host. The gateway would still be unable to determine if a packet addressed to 216.239.39.104 was meant for the public host (`google.com`) or a private host reusing the address in the server's network. With the traditional Layer 3 coordination, private reuse could be ruled out, but we would lose a key advantage.

This is not a defect of the approach nor of relaying. Connectionless protocols by definition do not retain a logical connection above Layer 3, so it does not make sense to try relaying them between unmodified Layer 3 end points in different realms. By the argument of representational closure (Section 1.6), only applications that present application level credentials should be permitted in any network. It is unfortunate that UDP has become used as the transport layer for numerous applications including the DNS itself and multimedia protocols including RTSP used here and RTP [SCFJ03, SC03], for want of this theoretical understanding. We would expect, in light of this thesis, that these will be revisited and revised along with most other IP protocols.

4.6 Measurements

Current network simulation platforms, like `ns2`, as well as research frameworks, such as Internet-2, have been designed to study lower level protocols and their performance, whereas the principal components of the present approach operate almost entirely as IP socket applications, i.e. at Layer 7, as described in the preceding sections. Since the deviation from current Internet architecture is principally in terms of the configuration effort required on large network scales, use of such a platform would not have helped meaningfully quantify the principal gain envisaged, viz elimination of global address space coordination and the resulting simplification and cost reductions. Though this intended longer term result thus remains unvalidated, a working prototype was clearly necessary to validate the workability and validity of the NAT inversion principle. This had been accomplished and demonstrated in 2001, as stated. The scenario of Fig. 4.2 has been recently resurrected using this unoptimized prototype to get measurements, which are found to be quite consistent with the expected lightness and efficiency.

4.6.1 Setup and localhost application first-connect times

The following measurements were performed on an IBM Netvista A22 desktop with 1.6 GHz Pentium 4 processor running Gentoo Linux 2.6.7-r8 kernel, running two ANS node processes A and B, node A connecting to B as child, as in Fig. 4.2 but on the same host and without the firewall. These were run with their TCP listening sockets bound to two different loopback addresses, 127.1.0.1 and 127.2.0.2, listening at RTSP ports 8554 and 9554, respectively, to demonstrably ensure separation of their packets.

The only real difficulty using multiple addresses in the loopback subnet on Linux is with ICMP: the loopback driver makes the source and destination addresses the same in the ICMP packet headers received over a raw socket, making it impossible to distinguish the packet sources and leading to routing loops when simulation of the scenario of Section 4.3 is attempted using loopback addresses on the same host. The problem does not occur with TCP or UDP, however.

Additionally, a first ANS “switch” node p to simulate a client host was run with root privileges so that it could bind to UDP port 53 in order to intercept and respond to the DNS queries from unmodified IP clients, as particularly described in Section 4.3 for ICMP. A second ANS “switch” node q simulated an application server host. On both, a pool of 10 IP consequtive addresses from the loopback subnet, starting at 127.10.1.0 and 127.20.1.0, respectively, were configured as the virtual address pool.

The procedure of Section 4.3 is followed with one difference in implementation: a TCP data link between the "switch" nodes is set up *a priori*, and the data channels are set up over this link by encapsulating with a packet header bearing only a virtual path (or connection) index and a checksum, similar to an MPLS label. This part had been originally implemented in 2001 partly to emulate ATM links and connections, since MPLS has an entirely different path set up procedure using a label distribution protocol (LDP) that depends on IP and would thus have conflicted with the present approach. The emulation of data links further served to limit firewall traversal to a single RTSP-established inter-ANS link and a corresponding inter-switch link, both using TCP to avoid having to establish new firewall "holes".

The measurements listed in Table 4.3 were then obtained as a baseline for comparing with connections over a network link, using a TCP client program `tsend` written to make and break TCP client connections repeatedly, with the following command line:

```
tsend -n times -[c|d|e] name_or_address:port
```

where `times` is an integer argument; option `c` specifies connect-only (for last argument given as numeric address so that no DNS lookup is involved); option `d` specifies DNS lookup only; and option `e` means "everything", i.e. both lookup and actual connection. The client code tears down every connection immediately, which frees the virtual IP address at the "switch" nodes. The switch code lacks any caching capability to reuse recently allocated addresses and associated virtual connections and to thereby reduce signalling and speed up response. This has served, in the past, to underscore the fact that the present approach does not depend on caching for its performance.

The application server was the basic TCP server `tcptalk`, with a boolean option (`-r`) to print the connecting client addresses in order to verify connection establishment. The first set (*direct connects*) were made without involving the ANS framework at all, by issuing the two commands to launch both the server and the client processes:

```
tcptalk -r 9999 > svr.log &
time tsend -n 100 -e 127.0.0.1:9999
```

The second set (*DNS lookup only*) used the client invocation

```
time tsend -n 100 -d localhost:9999
```

and the server log verified that no connections were made. Absence of DNS activity reports in the p switch log verified that the lookup was serviced from `/etc/hosts` by the client resolver, hence the observed latency of 0.15 ms is essentially that of this file lookup and parsing, plus the amortized overhead of process startup and shutdown and of the client execution. (The client does not call *malloc* or *new*.)

The third set (*ANS lookup only*) was obtained by invoking the following commands in separate windows to keep their output messages separate:

```
[win-1] rtspd -f etc/B.local    # ANS 'B'
[win-2] rtspd -f etc/A.local    # ANS 'A'
[win-3] swd -f etc/_p.cf        # switch 'p' (under sudo)
[win-4] swd -f etc/_q.cf        # switch 'q'
[win-5] tcptalk -r -s 127.10.0.1:9999
[win-6] cbind -h 127.10.0.1 -p 9086 /B/talk tcp 9999
[win-7] time tsend -n 100 -d talk.B:9999
```

The first two are the invocations for the ANS nodes, configured as dynamically loaded backends to to a base RTSP server as described in Section 4.1. The switch programs do not use RTSP. The additional option `-s` in the fifth command tells `tcptalk` to bind and listen on that address and port – the program otherwise treats the *address:port* syntax as a destination address to connect to and becomes a client instead of a server.

The sixth command invokes a shell script `cbind`, also already introduced in Section 4.3, that uses a sibling of `tcptalk` to inject a name binding request, at the local ANS protocol service port 9086 listened to by switch q, for the application server, while pretending to be from application server host 127.10.0.1. This results in the name binding *talk* being set up on node B, for a TCP server on 127.10.0.1, port 9999. The last command then causes the ANS signalling procedure of Section 4.3 being executed all the way till the return of the allocated virtual IP address to the client as response to its DNS query, *including the setup of the forward data path from the client all the way to the server*, involving a minimum of 11 IP hops $c{-}p{-}A{-}B{-}q{-}s{-}B{-}A{-}p{-}c$ (9 hops, c = client, s = server) and one hop link set up signalling (2 passes). The signalling is verified by the debug and log messages printed in the windows (and

	100 runs	*mean*
direct connects	0.111 s	1.11 ms
DNS lookup only	0.015 s	0.15 ms
ANS "lookup" only	0.041 s	0.41 ms
connects	0.508 s	5.08 ms
connects with debug	2.671 s	26.71 ms

Table 4.3: Localhost connection setup times

optionally to log files) in test runs, and the debug and log messages were suppressed for the actual measurement. The impact of the message output would be clear from the last pair of measurements, in the second of which the messages were unsuppressed.

4.6.2 Application first-connect times over the Internet

A second Linux host, a Dell C600 notebook with 1 GHz Pentium 3 processor running Debian Linux kernel 2.4.22 was set up to be the second host to simulate the scenario of Fig. 4.2 over a network. This second host was placed in a corporate network with high speed connections to the Internet, whose firewall permits inbound connections only via `ssh`. The first host, the desktop, similarly permits only `ssh` incoming connections from the Internet, to which it is connected via a high bandwidth cable modem.

The measurements in Table 4.4 were accordingly made by running the switch q process on the second host, configured to connect to ANS node B process on the first host via an ssh tunnel. Maximum compression was used on the ssh tunnel mainly to avoid impacting the corporate Internet links. The connect times without suppressing the debug and log messages were also collected and are shown for reference. Although the mean connect time rose from 51 ms to 126 ms, it is interesting that the user times actually decreased by 3 ms and system times rose by only 1 ms by using the ANS.

The decrease in the mean user time, which measures the CPU cycles incurred by the process in user mode, makes sense because the TCP connect handshake, which gets accounted to the client process, becomes confined to the localhost hop between the client process (`tsend`) and the client side switch p. Only the longer elapsed time accounts for the signalling between the switches p and q over the ssh tunnel, and the corresponding TCP connect handshake between switch q and the server (`tcptalk`).

	Time	#	Total	Mean	std.dev.
direct connects	real	500	25.326	0.0507	0.0951
	user	500	2.828	0.0057	0.0013
	sys	500	1.968	0.0039	0.0012
connects (no debug)	real	1087	137.486	0.1265	0.1679
	user	1087	5.785	0.0053	0.0016
	sys	1087	4.304	0.0040	0.0016
connects (with debug)	real	1601	223.613	0.1397	0.2483
	user	1601	8.465	0.0053	0.0019
	sys	1601	6.336	0.0040	0.0019

Table 4.4: Connection setup across Internet

Some increase in the mean system time was expected due to the addtional UDP DNS query incurred to look up the ANS name `talk.b`, because as the direct connects used the ssh tunnel, by referencing the 127.0.0.1 address from `/etc/hosts`, no DNS lookup was involved. The measured increase seems too little to account for this, but the decrease in the user time helps explain why: the system time for the TCP connect operation would have diminished along with the user time, but is more than made up by the increase due to the UDP exchange. The elapsed time is thus the best measure.

While the ANS seems to do worse than bare IP by $0.126/0.0507 \approx 2.5$, it includes name lookup, not accounted for in the direct connects: *name-routed ANS connections compete with IP connections excluding DNS resolution.* Demonstrations at INET'2001 and later have included subsecond name bind-connect-delete cycles impossible using the DNS (D-2, page 38). Indeed, the very fact that the DNS resolution is orthogonal to subsequent TCP connections suggests that the traditional approach conversely imposes a penalty for its better performance. The ANS does well because name string operations, *including parsing of HTTP-like connect and acknowledgement packets* (Table 4.2, page 52), cost much less than the network delays. The traditional approach improves over this but only by imposing the DNS query traffic, manual address space coordination and the promiscuity of Layer 3 transparency.

These values are as yet unsatisfactory because they do not adequately characterize the present approach for verifying its viability as a future Internet architecture. It would be necessary to further simulate appropriate traffic patterns, route discovery, changing topology, caching behaviour, etc. in a larger study for this purpose.

CHAPTER 5

A FORMAL THEORY OF NETWORKING

This chapter presents a formal treatment of various principles of the architecture, and explains the intuitions behind it.

Sections 5.1-5.3 establish key properties of the ANS that not only make it ideal for the IDP role but also argue against any other scheme in this role, even if we were to be content with higher administrative costs. Section 5.3 explains how the ANS exploits and represents the geometrical essence of the physical space bridged by the links, so that the ANS constitutes a natural coordinate reference frame already contained in any network graph, making express numbering of the nodes unnecessary. These ideas constitute a first step towards a computational notion of physical space, as opposed to the largely conceptual one of tapes in Turing's theory.

Section 5.4 presents the signalling protocol FAIR, formalizing the connection setup procedures described in Chapter 3. The signalling is independent of the route through the ANS, which would be non hierarchical if using LRTs.

Importantly, the ANS and LSPs are also independent of the realm infrastructures, needing at most local addressing and routing by the latter, and are hence inherently general and sufficient as a complete (inter)networking mechanism. The ANS by itself has all these properties, and LSPs primarily serve to improve the performance of flow routing. The IDP role specifically concerns ANS over Layer 3 (or Layer 2) datagram infrastructures, instead of a connection oriented "Layer 2.5" network like MPLS.

Section 5.5 discusses interoperability of LSPs with IP, i.e. for routing across IP realms without LSP encapsulation. Section 5.6 discusses the complementary case of using the DNS, in the place of ANS, for setting up inter-realm routes, particularly enabling, in Section 5.7, an architectural comparison with TRIAD and IPNL.

The treatment of ANS route discovery and of the use of LRTs in the routing of both datagrams and flows is given last in Section 5.8, since it leads to a sufficiency principle for networking beyond the limits of coordinated addressing of any kind.

5.1 Canonical principle of networking

The principal contention of this thesis, Principle 2 (on page 7), is formally justified in this section from a computational standpoint involving the properties of trees. A geometrical intuition is presented in Section 5.3. The principle is restated below for ease of reference.

Principle (Canonical principle of networking). *The least configuration needed for efficient network addressing and routing on any scale comprises local link interfaces and locally unique node names.*

Rationale. The notion of configuration intended in the principle is the information that needs to be supplied, typically in files and sometimes as command line arguments, for setting up a network usable by applications, i.e. with node addresses and routes. The rationale comprises several components established separately:

G-1. Minimization of the number of links to be configured, given by a *spanning tree*, discussed below as a lemma in its own right.

  Lemma 2 (Spanning tree property). *The fully connected graph of a given number of nodes with the least number of links is a tree.*

  Rationale. This is basic in graph theory and also the basis of the well known Spanning Tree Algorithm (SPT) in bridge routing, so the reader is best referred to a standard text like Ref. [Per99,p.58]. The idea is that in a connected graph of N nodes, some pairs of nodes can have more than one link between them, which would be redundant from the perspective of minimizing the configuration. When all such redundant links are removed, the result is a spanning tree. $\diamond$

G-2. Sufficiency of a spanning tree to itself serve as an *addressing and routing tree*, for which a formal construction of addresses and routing is given below. Only local uniqueness of the node labels is required for this construction, in accordance with the principle, and it serves for both datagram delivery and setup signalling for connections or flows, as shown by detailed examples in Chapters 3 and 4.

To be efficient, the address construction should entail little or no computational cost. This condition would not, for instance, favour the current computational schemes for synthesizing addresses in sensor networks. The efficiency of routing must be likewise considered from all three standpoints of per-hop forwarding, route lengths and route discovery, together with reasonable assurance of robust routing, viz that all destinations physically connected in the network should be actually reachable by the applications.

Efficient address construction and robust default routing, which does not depend on discovery and is available instantly, are both established in Section 5.2 as spanning tree properties consistent with the present approach. Specifically, the address construction involves no computation beyond the link setup and child registration, as in the prior examples, and only locally unique labels, which are invariably available to or readily generated by the deployment procedures.

As the principle concerns minimization of configuration, the only reason to consider the discovery of non-tree routes is to improve the efficiency of routes in terms of overall lengths and costs. As the name trees would likely be aligned with provider hierarchies in the simple cases as described in the examples, the default routes would be already fairly efficient, and comparable to current inter-domain routing, as well as to NIRA. In more complex deployments involving secondary ANS trees (dANS), translation to provider-aligned name paths would be again implicitly available. It is therefore meaningful to treat route discovery in general as optional over robust and already modestly efficient default routing. Note also that while NIRA has similar default routing, the address construction needs to be coordinated and not free.

Further, given the general availability of default routes virtually for free, the only reasons to configure more links would be to handle additional traffic volume or to exploit business opportunities to deploy alternate or cheaper routes. Since all such routes would at most comprise additional links between the tree nodes, the address construction would be unaltered. With route discovery, knowledge of these routes becomes available in routing without their manual configuration.

Thus, the inclusion of non-default routes and their discovery does not materially alter the contention of the principle. Formal analysis of the address construction, route discovery and routing of both signalling and flows over an arbitrary mix of default and discovered routes, given in Section 5.8, substantiates this view.

G-3. Decoupling of actual transport addresses, at Layers 2 as well as 3, surrounding each node or link end point, from the overall end to end addresses and routes given by the formal construction. This has been stated as a lemma and shown to hold by example in Section 3.4, along with a precise analysis, in Section 3.1, of the converse constraint in NAT.

G-4. Sufficiency of these procedures for an indefinite number of nodes. Adequacy for indefinitely large number of nodes each capable of handling only a finite number of links is once again a basic property of the tree structure, and the reason for its widespread use in diverse forms for representing and manipulating data.

Though the present approach merely reuses this known property for networking, the result is such a break from current notions of network addressing, such as represented by IPv6 and alternatives proposed hitherto (see Chapter 1), that it merits attention as a principle in its own right. It was hence stated as Principle 1, and is justified separately in Section 5.6.

The sufficiency for very small scale networks has been demonstrated with two and three node configurations in Chapters 3 and 4, and the links in these working examples can be trivially replaced with dedicated lines, such as serial interfaces and cables. (The earliest experiments were in fact done on an experimental ATM network at IBM Research using raw ATM virtual circuits, and the prototype code is designed with a C++ class structure directly representing this general notion of interfaces and links.)

The principle is therefore valid on any scale whatsoever, as stated.

The tree-based address construction and routing, to complete this justification of the canonical networking principle (G-2), is discussed next, as mentioned. ◇

5.2 Canonical addressing and routing theorem

In a network comprising a set of nodes S and an unspecified set of links, the *address* of a node $s \in S$ must be a *communicable* reference to that node, as it would be useless otherwise. This definition incorporates RP by requiring addresses to be high level representations, appropriate for definition and use at the application layer, and includes the transparency requirement [Car00]. Hence the results should obviate any existing need to seek transparency all over again from Layer 3.

For this choice to work, the address should be *representable*, reflecting the RP, by concatenating symbols from a communication alphabet Σ (as in Shannon's theory), so that we may write $addr(s) \in \Sigma^*$, and express the *addressing function* as an injective mapping $addr : S \to \Sigma^*$.

This definition is too general to permit a simple mathematical inversion to uniquely identify a node s from its address $addr(s)$, and such an inversion would be meaningless without an *a priori* or an independent *routing function* to actually take us to node s from any other location $t \in S$. This overly general routing notion is what the DNS, with its resolution function in the role of inversion, embodies, for example, since the returned IP address is currently regarded as a location identifier. At least one class of addressing functions exists, so says the principle, with zero global coordination cost and yet sufficient to provide routing. To identify this class, we begin with a working definition of a routing function, and then consider its representational requirements.

A routing function $\rho(.,.)$ is a "relative inverse" to addressing, as, given a starting node $t \in S$ and an address $addr(s)$, it must produce a path, i.e. a continuous chain of links from t to s. The basic constraints on the addressing function would be that

- an unlimited number of nodes be addressable and reachable, even if each node is physically capable of supporting only a limited number of links; and that
- the addresses be absolute, i.e. independent of the starting location.

The first constraint acknowledges reality, and requires support for relaying, or forwarding, of packets from node to node i.e. routing. This requirement is less trivial than it would seem. For example, unlimited direct linkage is assumed and is necessary in the DNS, for example, as the name resolution process is defined to begin, whenever not satisfied by the client nameserver cache, directly from the root label by looking up one of the thirteen public root nameservers, to be then successively redirected to an authoritative nameserver for the target domain. This requires the client server to directly connect to the root and to each of the succession of authoritative servers. This direct linkage is provided, of course, by IP, but at the cost of manual configuration. (The premise of direct linkage by IP also plays an optimization role in DNS design, as otherwise, the resolution process would have had to use an upward tree-walk from the client server through authoritative servers for successively larger encompassing domains, possibly all the way to the root. At every step of the upward walk, however, the search would needlessly include subdomains that had already failed.)

The first constraint would be satisfied by using a spanning tree, since, as remarked in the previous section (item G-4), connecting an unlimited number of nodes each with a limited capacity for links is a basic tree property.

The second constraint requires that there be at least one node, to be designated the root and denoted by the familiar symbol '`/`' $\in \Sigma$, that can be used as a reference to construct the absolute addresses. By the first condition, for each pair of non-root nodes s and t, there must also be the route $\rho(s,t) = \rho(s, root).\rho(root, t)$, where $\rho(s,t)$ is short for $\rho(s, addr(t))$ and '.' denotes relaying. The canonical address construction and routing are formalized by the following theorem.

Theorem 1 (Canonical addressing and routing). *In any tree of nodes S with each node $x \in S$ having a locally unique label, denoted $\lambda(x) \in \Sigma^*$,*

- *an addressing function $addr(.)$ exists which is complete over S, that is, for every node $s \in S$, $addr(s)$ is uniquely defined relative to a designated root node; and*
- *the routing decision at each node r along a route $\rho(s,t)$, i.e. $r \in \rho(s,t)$, needs to consume at most one next symbol from the destination address $addr(t)$ matching an adjacent node label.*

Proof. Since the theorem is about existence, we can use the hindsight of the previous chapters to directly define the addressing function, and then verify that it satisfies both properties. We define it as follows:

- $addr(root) = \lambda(root) \equiv \phi$;

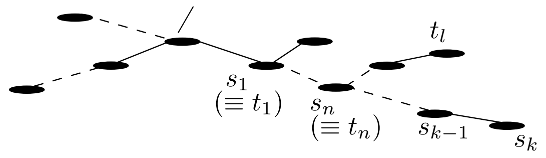

Fig. 5.1: Canonical addressing

- for every other node $s \in S$, if $\rho(root, s) = root.s_1....s_{k-1}.s$ denotes the tree-walk route from the root to node s, then $addr(s) = \lambda(root)/\lambda(s_1)/.../\lambda(s) \equiv /\lambda(s_1)/.../\lambda(s)$.

The completeness and uniqueness of $addr(.)$ follow trivially from those of $\rho(root, .)$.

For the second part, consider that the address for a destination t would be given by $addr(t)$ similarly corresponding to $\rho(root, t)$. Denoting $\rho(s, root) = s.s_{k-1}...s_1.root$ and $\rho(t, root) = t.t_{l-1}...t_1.root$, there must clearly be an integer n in the range $1 \leq n \leq \min(k, l)$ such that for all integers i such that $1 \leq i \leq n$, we would have $s_i = t_i$:

- if $n = \min(k, l)$, one of s or t must be an ancestor of the other in the tree;
- if $n = 1$, the only node common to $\rho(s, root)$ and $\rho(t, root)$ would be root;
- for all n between these values, there would be a common set of ancestors leading up to the root (Fig. 5.1).

In the third case, we would have the shorter route $\rho(s, t) = s.s_{k-1}...s_n.t_{n+1}...t_{l-1}.t$ skipping $2n$ links to the root and back by turning at a branch point $s_n \equiv t_n$. For $n = 1$, the only branch point is the root itself, so that $\rho(s, t) = s.s_{k-1}...s_1.root.t_1....t_{l-1}.t$. For $n = \min(k, l)$, the route reduces to $\rho(s, t) = s.s_{k-1}....s_n$ when $\min(k, l) = l$ and to $\rho(s, t) = t_n...t_{l-1}.t$ when $\min(k, l) = k$. The forwarding decision is required, at each node $r \in \rho(s, t)$, to determine the next node.

Consider first the general case, $\rho(s, t) = s.s_{k-1}...s_n.t_{n+1}...t_{l-1}.t$. At s, the routing decision needs to identify s_{k-1} as the next node, from the given address $addr(t) = /\lambda(s_1)/.../\lambda(s_n)/\lambda(t_{n+1})/.../\lambda(t_{l-1})/\lambda(t)$. Since the prefix $/\lambda(s_1)/.../\lambda(s_n)$ exactly matches the first n labels of $addr(s)$, the routing decision must be to forward towards the root for a count of $k - n$ hops to reach s_n. This can be arranged in one of at least two ways, by reexamining the prefix at each hop in the ascent to s_n, or by setting an "ascent hop count" in the packet header, which is to be decremented at each hop, stopping when the count becomes zero.

At s_n, the prefix may be dispensed with altogether, or a "remaining path index" set in the packet header to the remaining address segment $\lambda(t_{n+1})/.../\lambda(t_{l-1})/\lambda(t)$. At s_n, the routing algorithm must match $\lambda(t_{n+1})$ to the next hop node t_{n+1}, and subsequently at t_{n+1}, it needs to match $\lambda(t_{n+2})$ to node t_{n+2}, and so on, eventually matching $\lambda(t_{l-1})$ to node t_{l-1}, and $\lambda(t)$ to node t_l, to arrive at t. These matched labels clearly correspond to successive labels in the address segment at s_n, and are directly the name labels of the next node at each hop from s_n, so that they can be successively consumed, or the remaining path index incremented, at each hop.

In the case of $n = 1$, the ascent count would be initially set equal to the depth of s in the tree, and the procedure is otherwise unchanged. In the case of $n = \min(k, l)$, if $\min(k, l) = l$, t would be an ancestor of s, the ascent count would be initialized to $k - n$, and t would be reached when the count decrements to zero. If $\min(k, l) = k$, on the other hand, t would be a descendent of s, the ascent count would remain at zero, and the routing would only involve consuming labels after deleting the prefix $addr(s)$ from $addr(t)$. $\square$

5.3 The reference frame perspective

The "physicists' approach" presented below exploits the observation that the routing decisions incurred in the physical routing of a packet as such trace out a distributed decision tree, in which each node is a physically separate forwarding node and operates, in a well aggregated hierarchical system, on a different segment of the address.

Conversely, simple concatenation of node labels along such a routing decision tree should suffice for addressing, with no requirement that an individual node's identity be unique other than at its preceding node. This is the concept of *relative addressing*, as historically used in UUCP and ATM PNNI [ATM94], supported in IP as source(-specified) routing, and related to label switched routing in MPLS [Per99,§7.1].

The further step in the present architecture, of *canonical addressing*, reflects two additional observations, the first being that the only basic flaw of relative addressing is the absence of *absolute addresses* to permit use from anywhere in the network to uniquely identify destinations. The second is that this flaw really had little to do with the relativeness of the individual node names, and more with the fact that the prior relative addressing schemes did not select a hierarchy or a reference point as origin for a networking reference frame. As shown next, a hierarchical routing topology is itself a natural system of coordinates, and the key advantage of a coordinate system, in physics or geometry, is that *when given a coordinate frame, spatial elements do not need to be expressly numbered!*

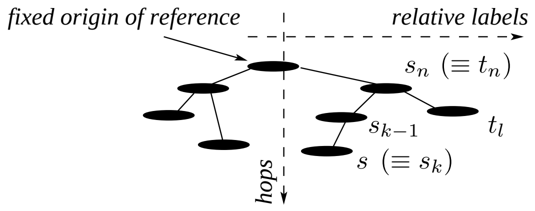

Fig. 5.2: ANS construction as a natural coordinate system

Fig. 5.2 explains this intuition. If the hierarchy were one-dimensional, i.e. without branching, all the nodes would be strung out in a line and any node would be directly addressable by its position in this line, so that the address constructed in Section 5.1, $/s_1/.../s_{k-1}/s$, would reduce to the index k, counting from the root. The difference made by branching is that name labels are needed to distinguish, at each hop on the downward walk, between the immediate descendents, and hence must be unique, but only at the immediate node. The branch point ending the initial ascent is determined by the common prefix between the starting node's own address and the destination's, but the prefix computation again compares the labels only between matching positions starting from the left. Relative labelling is thus generally adequate and the addresses form a two-dimensional coordinate system, as shown.

There are clearly four requirements for the construction of such an address space:

◇ the nodes need to be linked in a hierarchy;

◇ the nodes must be given locally distinct labels;

- ◇ each node must be able to associate a descendent's label with a connecting link (for downward routing); and
- ◇ each node must know its address (for initial ascent).

The last two requirements can be efficiently ensured by a registration procedure when the link is established, in which the child negotiates for a preferred label and the parent responds with its own address. The child can immediately compute its own address by simply appending the delimiter '`/`' and its own label confirmed by the parent. The second requirement can be efficiently ensured in the label negotiation in various ways, e.g. child presenting a small list of preferred labels, parent suggesting an alternative label, appending an incrementing number, etc. using no more than $O(1)$ messages.

The main problems are those of picking labels that would be convenient to users and applications, and linking the nodes in a hierarchical fashion. This is admittedly not trivial in the context of *ad hoc* networking because

- ▷ the child nodes may have to generate labels before knowing which parent they might connect to, and the labels would then have to be separately matched up application level identifiers;
- ▷ the link order would not be known *a priori*, hence automatic linking would very likely lead to loops instead of a hierarchy; and
- ▷ there can be no assurance, without *some* global coordination measure, that the resulting structure would even remotely agree with user or application needs.

However, these problems are unlikely to arise when the nodes are labelled and the links set up manually, which is necessarily the case for application hosts, as well as for most routers and realm gateways, as each of them, and the corresponding physical links, involves physical installation and basic initialization, including some labelling, for the user's or administrator's own reference, even without enabling IP.

Therefore, though we cannot eliminate *all* configuration, which would correspond to the *ad hoc* problem, we *can* make opportunistic use of the "pre-networking" setup that to obtain the network addressing and routing automatically. In the prototype implementation described in Section 4.1, a simple one-step accept/reject registration scheme was used with this view.

In true *ad hoc* scenarios, such as remotely deployed sensor or battlefield networks, we would not really care about the precise values of the labels any more than we do currently about the automatically assigned addresses, making a Layer 3 numbering scheme, i.e. IP, appropriate in this case. The application level topology, even if just a flat space of names, becomes all the more relevant in the high level identifier role, and the name space has to be a distributed, cached database, like the DNS.

In the non *ad hoc* scenarios, however, we do invariably set labels that are directly or indirectly representative of the links, and hence usable as coordinates. In Fig. 5.2, for example, the labels s_k and s_{k-1} are indirectly representative of their parent-child link – one might think of the '`/`' delimiter as a wildcard for such links.

In a general, existing network, we would be faced with another problem: selecting the links to use for the hierarchy. This is however an *a posteriori* problem in which we would have already put in more configuration effort, for more links than needed for a single (spanning) tree. The principle only claims that we could save configuration effort when setting up a new network or overlay. While any spanning tree could be used as a coordinate system, effort can be saved only while building anew.

The utility of the principle is that the construction can be also used for first-time overlays between logically unconnected networks (or realms or domains), to obtain corresponding multi-network (or multi-realm or multi-domain) addressing, without global coordination at any level. From this standpoint, the entire problem of Internet addressing, from the initial deliberations of Shoch and others leading to IPv4 to later schemes including IPv6 and its alternatives as discussed in Chapter 1, have been a placeholder for this basic and more powerful coordinate system approach.

5.4 Generalized flow setup

Fig. 5.3 depicts the simplest case of a sequence of ANS hops, as a first step in the generalization of the path setup procedure introduced in Chapter 3. No distinction is made between the canonical (hierarchical) and non-canonical (lateral or transverse) links, i.e. using LRTs, in the subfigures. Hence the conclusions will be valid whether or not the ANS route is canonical, i.e. strictly follows the tree-walk of Section 5.1, so long as all nodes are ANS nodes.

As the ANS is a routing network, the forwarding states are needed only for flows and connection-oriented protocols. Each state, looking either way as indicated in the figure, would be an LSP label containing the tuple $\langle next\_ans\,, label\rangle$, where $next\_ans$ is the ANS address of the next node and *label* identifies the connection or flow, as in MPLS. The number of hops is in principle unlimited. The number of forwarding states per node, corresponding to the *density* ρ of connections or flows in the network, has a logarithmic lower bound $O(\log_k \rho)$, when recursive aggregations are employed, as will be discussed in Chapter 6. The basic signalling procedure for setting up the LSP, called Forward path setup with Acknowledgement, Inband signalling for Reverse path (FAIR), is reviewed next for completeness of the present treatment, and will be formally treated in Section 5.8.

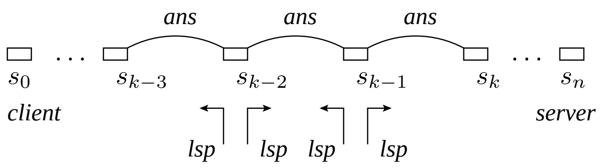

Fig. 5.3: Segments in an extended path

FAIR begins when a connection or flow setup request from a first node s_0 arrives at the destination node s_n and is delivered to a `listen`ing application process. If the process `accept`s the connection, a new entry is allocated in the OS socket table with inbound routing pointing to the handle returned to the process by the `accept` call. An index λ_n to this socket is then sent out, directly from the OS "protocol stack", as acknowledgement to the preceding node s_{n-1}, along with the received request id or number and the return (ANS) address (of s_0). Node s_{n-1} then has the information to set up a forwarding table entry of the form $\langle addr(s_n)\,, \lambda_n\rangle$.

Based on the return address, s_{n-1} then forwards the acknowledgement on to s_{n-2}, replacing the index from s_n with its own index to the new entry, λ_{n-1} say, enabling the next node s_{n-2} to set up its table entry $\langle addr(s_{n-1})\,, \lambda_{n-1}\rangle$. The process is repeated until s_0 receives the acknowledgement from s_1, with the corresponding index λ_1 and the original request id; s_0's host OS now tries to reconcile the acknowledgement with request. If the requesting process still awaits, a new socket table entry is created (as in TCP `connect`), with the outbound information $\langle addr(s_1)\,, \lambda_1\rangle$. A handle to this entry is returned to the process.

This merely completes the forward path from the client s_0 to a server s_n, but it is end to end and can be used for efficiently setting up a reverse data path, along the same route, by sending an inband control packet, together with an index η_0 to (the receiving side of) the new socket entry. At each successive node s_k,

- the preceding node s_{k-1} is identified, by the arriving link or "from" address;
- the forward path index λ_{k-1} is looked up to retrieve the onward index λ_k and the next node address $addr(s_{k+1})$;
- a new table entry, of index η_k, is allocated, and is initialized with the tuple $\langle addr(s_{k-1})\,, \eta_{k-1}\rangle$; and
- the new index η_k and the retrieved index λ_k are inserted in the control packet, replacing the previous values in these fields, and the packet is then forwarded to the next node s_{k+1} addressed by the retrieved forward path entry.

When the control packet arrives at the destination s_n, its OS adds the return path information $\langle addr(s_{n-1})\,, \eta_{n-1}\rangle$ to the incomplete socket table entry corresponding to index λ_n, and can now dispatch already queued packets if any to s_0. The forward path may be also used by the OS at s_0 to initiate teardown if the client dies.

5.5 Generalized asymmetric use

Fig. 5.4 shows the various asymmetric use scenarios that need to be considered for a practical deployment of the present approach. Fig. 5.4(a) is the basic ANS multihop scenario of Fig. 5.3 for comparison. Layer 4 forwarding is assumed (see Section 3.4.1).

Fig. 5.4(b) describes the basic case of an IP client s_0 within an IP network, that does a DNS lookup to get to the application server s_n in a foreign realm, as described in Sections 3.4 through 3.6. The forwarding state looking inward into s_0's (IP) realm from the next node s_1 would be a VAS address state corresponding to an LSP state looking onwards towards s_2, both set up, as in Chapter 3, when the onward ANS hops

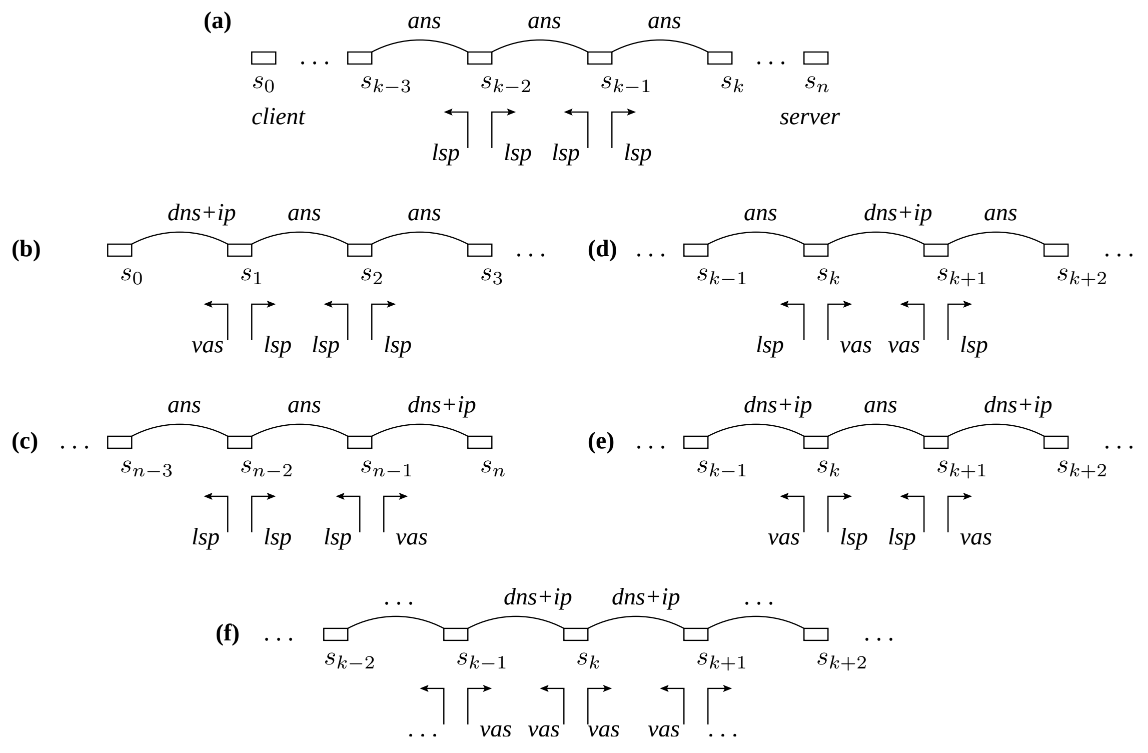

Fig. 5.4: Generalized paths with Layer 3 segments

$s_1 ... s_n$ return the onward index λ_2 with the returning acknowledgement, as described the previous section.

Fig. 5.4(c) describes the converse case of an IP *server* s_n hosted in an IP network. In this case, we would need the previous ANS node s_{n-1} to look up the realm address of s_n possibly from the realm DNS, open a client socket to s_n as described in Section 3.5 and in Sections 4.2 and 4.3, then allocate an LSP state, of index λ_{n-1}, linking to this onward socket, and return the index λ_{n-1} as the next hop information back to the preceding node s_{n-2}, as before in Section 5.4.

Fig. 5.4(d) depicts the first scenario not already treated in the preceding sections, that of an intermediate realm over which, for some reason, we need to use IP routing without LSP encapsulation. A combination of the preceding cases suffices for the path construction from s_k to s_0, by first applying the procedure of Fig. 5.4(b) at node s_k as if it were the final client host. This results in a client socket at s_k, for which we now need to allocate an LSP state of index λ_k, link this state to the socket, and return the index λ_k to the preceding node s_{k-1}, following the procedure of Fig. 5.4(c). The catch is that at s_k, the onward socket connection must be made to a virtual address allocated at s_{k+1}, whose address and port information is needed at s_k for making the connection. Accordingly, node s_{k+1} opens a *server* (`listen`ing) socket bound to this allocated address, to receive the client socket connection, and only then forwards the acknowledgement packet to s_k with the address and port information.

Fig. 5.4(e) depicts the converse of the Fig. 5.4(d) scenario, comprising an ANS link sandwiched between IP routing realms. In this case, a client socket is opened at s_{k+1}, and a corresponding LSP state index λ_{k+1} returned to node s_k. At s_k, a VAS address is first allocated, then a server (`listen`ing) socket is opened bound to this allocated address, as in the preceding scenario, and the rest of the return path is treated exactly as if s_k were the application server of Fig. 5.4(c), but with the VAS address and port information given by s_k.

Lastly, Fig. 5.4(f) depicts the natural generalization from Figs. 5.4(d) and 5.4(e) to paths requiring more than one consecutive unencapsulated IP hops. The key step in the handling of this case is at a typical node s_k linking two such hops, that differs from the preceding cases in not involving an LSP state. Instead, two VAS addresses are allocated, one on each side, an onward client socket is opened towards s_{k+1} with the address from the onwards VAS pool, based on $addr(s_{k+1})$ and the (protocol-) port number information obtained from s_{k+1}, and then a server socket is opened bound to the address from the preceding hop pool, as in the previous two scenarios. Forwarding is then set up between the two sockets, and the address, (protocol and) port information of this server socket is passed onwards to the preceding node s_{k-1}, as in the preceding scenarios.

The process changes somewhat with Layer 3 forwarding, but the differences are straightforward and uninteresting in terms of the architectural purposes of this thesis. A more pertinent change of circumstance, considered in the next section, is the use of DNS, instead of ANS, as IDP, which would be particularly applicable over a network of IP realms. This is complementary to the cases of unencapsulated Layer 3 hops just considered, where the IDP was still assumed to be ANS.

5.6 Rationale of unlimited inter-networking

We can now justify Principle 1 from on page 7, reproduced below for convenience.

Principle (Unlimited inter-network). *A necessary and sufficient condition for unlimited effective addressing using fixed length numeric addresses is that inter-realm routing be done via names.*

Rationale. The sufficiency has been already established by the preceding chapters based on using the ANS as IDP over an unlimited number of unrelated Layer 3 realms, since the ANS is a naming overlay as required by the statement of the principle, and inter-realm coordination of Layer 3 (numeric) addressing was expressly dispensed with, notably by allowing for local reuse of public addresses in Chapter 3.

The necessity is more interesting. The first is necessary since an infinite Layer 3 address space would require indefinitely long addresses. Variable length numbering has been proposed as an alternative, especially as it is commonly used in telephone systems. However, any global Layer 3 (or numeric) address space needs to be closely coordinated, and this in the more dynamic setting of Internet addressing tends to devolve into fixed length schemes, as observed by Deering [Dee00][1], and thus gets limited to finite realms.

The necessity of the second condition follows partly from the first, since in absence of numeric (or Layer 3) addressing across realms, some form of an addressing overlay is necessary, which has to take on some corresponding responsibility for inter-realm routing. The overlay could be obtained as another layer of numeric addresses, but that would merely make the resulting space a direct product space of the realm and inter-realm addresses. To see that necessarily holds, consider that a finite set of realm addresses may be denoted as the set $X \subset I$, the set of all integers. If we use a second finite set of numbers $Y \subset I$ to distinguish realms, the total address space becomes simply $Y \times X$, and is again finite, and the argument can be applied inductively to any finite number of layers. Thus, there is no way to obtain unlimited numeric addressing by any finite number of layers of indexing.

1. Referring to CLNP, which was initially intended to support variable length addresses.

The point of the above principle is however that any such exercise, including IPNL, must be fundamentally unnecessary because at the highest level, there would be a name space already operating across the realms, and the logistical constraints that reduce numeric addressing to fixed lengths do not hold for names. More specifically, names are not coordinated globally like an address space, but locally by parent-child negotiations, to which Deering's observation does not apply.

Now consider that there can be only two basic kinds of destination identifiers: those that can be interpreted directly as a logical route, like those of the bANS or the UUCP "source-routes", and those that translate into another addressing system, like those of the DNS or a dANS. In the latter case, the names and addresses both identify foreign destinations, making them functionally redundant.

To determine the necessary component, we must try to eliminate either the names or the translated addresses from the required role of inter-realm identifiers. One could conceivably restrict addresses to finite realms, reducing to the bANS case, but there is fundamentally no way to eliminate names from this role except in a very small set of application scenarios, a conclusion already stated formally as the representational principle (RP). Hence, names are necessary, and sufficient, as multi-realm identifiers for an unlimited effective Layer 3 network. □

It is interesting to examine how one could use the DNS as an IDP, since this too is assured by the principle. Consider again the scenario of multiple DNS-IP hops as shown in Fig. 5.4(f). The DNS query from s_{k-1} would need to first reach node s_k, and then be forwarded to s_{k+1} as an onward DNS query, on the outbound pass. For the query to arrive at s_k, it is necessary to define s_k, in the DNS of the intervening realm (between s_{k-1} and s_k), as the authoritative DNS server for a suitable subdomain of `.ans` that includes the application server host s_n, and similarly, s_{k+1} as authoritative in the next realm for the same foreign subdomain. The protocol and port information needs to be returned along with the VAS addresses in the returning DNS responses, and this can be done preferably by adding a new, extended address record type that includes fields for this additional information.

The real obstacle lies in the resolver API used by *socket* applications: the resolver functions invariably return the *hostent_t* structure which has no fields defined for protocol or port (D-1, on page 38). These parameters have been traditionally regarded as *a priori* knowledge, to be hard-coded or looked up in `/etc/services`, which is distributed with Unix operating systems. In hindsight of the present thesis and the above principle, it would make sense to extend the DNS reply format to include at least the port information.

5.7 Comparison with TRIAD and IPNL

It is interesting to compare the above "DNS-only version" of the present approach discussed in the previous section against TRIAD and IPNL.

TRIAD introduced a widearea relay addressing protocol (WRAP) for setting up relay states like those of the scenario of Fig. 5.4(f), but that is where the similarity ends. TRIAD poses rather than solve the naming problem, to quote "good directory service is needed" [Che99], and its solution amounts to a DNS (or an Active Directory) literally wrapping over multi-realm source routes, called Internet relay paths (IRPs). In turn, the IRPs are rather like UUCP, in being entirely relative, and like IP source routes or the designated transit lists (DTLs) in ATM PNNI, both involving absolute node addresses. The TRIAD notion of active directories resembles the Nimrod-FARA notion of dynamic mapping, and the implementational ideas of DHTs [BLR+04]. As a result, TRIAD has irreparably deviated from its original goals of unlimited effective addressing and name-based inter-realm networking, and offers no useful conclusion.

IPNL, on the other hand, envisages use of the DNS even on the multi-realm scale, and includes a detailed consideration for setting up zone table entries for resources in foreign realms. The main difference, which is illustrative of the basic difference in perspective, is that IPNL allows for caching of foreign nameserver data, by tagging the imported items with a dynamically managed foreign realm id. IPNL avoids relaying by constructing a larger address format including the foreign realm id; these larger addresses are returned by the DNS query for foreign resources, and require protocol stack shims to be handled at both the client hosts and the realm gateways. This is still a lot of effort for the limited principal advantage, if any, over TRIAD and the present approach, of directly transporting Layer 3 datagrams.

This advantage is challenged by the contention of Section 1.1 that Layer 3 routing transparency is a fundamentally bad idea, and that all datagram applications should be required to negotiate security and authentication *inband* like TCP, via an IDP. Lest it seem that a combination of IPNL and the present approach might be somehow used, notice further that IPNL's DNS caching support favours (unauthenticated) Layer 3 transparency by delivering IP datagrams – without inter-realm name lookups and re-computation of the realm ids.

The DNS-based version of the present approach would still have explicit namespace routing, and would not be a totally bad idea because NS records for the next realm nodes would get cached, and the NS record caching is the key to DNS scalability and performance [JSBM01]. Name routing is the only way to ensure "socket-by-socket" authentication, but involves redoing foreign realm ids, or, simply, setting up LSPs.

5.8 Discovery and routing in the large scale limit

This section presents a formal treatment of ANS construction and routing involving route discovery and discovered routes represented in LRTs, as well as routing on very large scales, where the discovery may be often incomplete. This section is taken from the journal paper, as mentioned, and revisits several results of the previous chapters from an algorithmic perspective.

Section 5.8.1 introduces a formal notation of ANS construction and routing for this purpose. It also serves to establish the general simplicity, cheapness and autonomic addressing features of the ANS by showing that the complexity of a handshake for adding a node is $O(1)$ messages, and that of building a spanning tree for addressing and routing *is exactly* 0 as the namespace is itself a spanning tree – this incidentally beats the bridge spanning tree protocol with respect to Local Area Networks (LANs), i.e. Layer 2 construction, as the bridge protocol does require computation [Per99,§3.3], once again demonstrating the power of RP and of the present canonical approach.

Section 5.8.2 extends the notation to include non tree (lateral) links and to IP-like datagram routing using both sets of links, essentially correcting the triangular routing inefficiency of the default (canonical) routing. It also reveals a remarkable opportunity provided by the latter, viz as default routes are guaranteed by the name space tree, *packets can now be default-routed upon route cache misses without dropping them or delaying other traffic*, unlike the situation with caching in host memory architectures or current inter-domain routing. The utility of this option remains to be investigated by statistically significant simulation of traffic patterns and link failures.

Section 5.8.3 extends the notation to include path setup and virtual path routing for connections or "flows". Flow routing is specially provided for in IPv6 [RCCD02], partly, it would appear, as a concession to the reality of MPLS and its performance over traditional packet routing (cf. [NLM96], treating flow routing of IPv4 over ATM). Connection ("flow") oriented routing would be necessary in the present approach for the further reason, in the purely theoretical consideration of unlimited multirealm scales, that very long ANS addresses in the packet headers would again make simple packet forwarding inefficient. Syntactic transformations can be employed to shorten addresses in the headers, such as replacing the upward ANS hops with the filesystem "../" notation and runlength encoding thereof, but a more general solution would be clearly desirable and is elegantly provided by label switching or virtual path routing.

A further problem on large scales, which must be addressed for the theoretically unlimited multirealm scale, is the fundamental physical limit on route discovery, as the exploration all possible paths, or even a substantial set of alternative paths, would necessarily entail a corresponding number of network hop delays. This is why in the current Internet, shortest path routing protocols are only employed within subscriber networks and the "exterior" routing is only of "best-effort" quality using BGP.

The present architecture provides a remarkable solution for this "infinite scale", viz *guided routing*, that is, routing of both packets and path setup along the namespace-defined routes given by the base (bANS) network or a secondary name tree (dANS). As the ANS construction and routing are both "everywhere local", they remain feasible on indefinitely large scales, well beyond the practical limits of self-coordination [For03] or optimal structuring [KRS02] schemes for unmanaged networks, and indefinitely past the breakdown of manual global coordination. It will be shown in Section 5.8.3 that the ANS provides a sense of direction as a soft degradation to canonical routing on very large scales.

5.8.1 Complexity of canonical addressing and routing

The ANS construction is characterized below in terms of point-to-point links, denoted $link(x, y)$, where x and y represent nodes and each link supports a *reliable* two-way messaging protocol like SSCOP in ATM signalling [ATM02] or TCP, as used in the prototype system, as described in Chapter 4, but without assuming the node addresses to be global and coordinated within the ANS. First some definitions.

Each node x in an ANS S has access to a local *real address* (RA) space, comprising the set $links(x) = \{y \mid y \in S \wedge \exists\, link(x, y)\}$. A global RA exists iff $\exists\, link(x, y)\; \forall\, x, y \in S$, or equivalently, $links(x) = S\; \forall\, x \in S$, i.e. if S is fully connected. For a node $x \in S$, its parent node, together with the connecting link from x will be denoted $par_S(x)$.

The effective address (EA) of x with respect to S, which is to be determined, will be denoted $addr_S(x)$. Therefore, $par_S(x) = null$ iff x is the root node, $root(S)$. The children of node x form the set $children_S(x) = \{y \mid par_S(y) = x\}$, stored at x as a table, each entry comprising a child's name and the link thereto.

For nodes $x, y \in S$, y would be reachable in S from x, notated $reachable_S(x, y)$, iff $par_S(x) = y$ or $par_S(y) = x$ or $\exists$ a route $P(x, y) = \{x, s_1, \ldots, s_k, y\}$ such that $reachable_S(x, s_1)$, $reachable_S(s_k, y)$, and $\forall\, i = 1 \ldots k-1,\;\; reachable_S(s_i, s_{i+1})$. The length of route $P(x, y)$, written $|P|$, is 1 hop if x and y are directly related, and $k+1$ hops otherwise. The efficiency of construction will now be established.

Lemma 3 (Efficient EA). *$addr_S(x)$ can be acquired in $O(1)$ messages.*

Proof. The construction is given in Algorithm 1 below; note that the first test, if S is empty, does not require messaging. The addition of x to S in the last step is again conceptual. Messages are involved only when there is a parent. The first message is a registration request from child x to parent y, carrying an optional identifier $name(x)$ chosen by x, that y verifies to be locally unique, that is, no other child of y is registered with the $name(x)$. Recent names may be disallowed as well to prevent confusion and inadvertent theft of identity. If $name(x)$ is locally unique, the parent accepts x as a child and returns its own EA $addr_S(y)$ in acknowledgement, enabling x to compute its EA by prefixing $addr_S(y)$ to $name(x)$. Otherwise, y may synthesize a unique name and return it with the acknowledgement, preserving the bound. Another possibility having the same bound is for x to send a list of preferred names for y to choose from. Even if multiple iterations are allowed, the number of messages required is governed by local uniqueness at y, hence the bound is independent of the network size. □ The next lemma characterizes canonical routing as formally introduced in Section 5.2.

Algorithm 1 Incremental construction of EA

```
**Require:** network $S$; new node $x$
**Ensure:** registration of $x$ in $S$; efficient computation of $addr_S(x)$
  **if** $S = \{\}$ **then**
    $par_S(x) := null$; $addr_S(x) :=$ '/'$\cdot name(x)$; $S := \{x\}$
  **else**
    select node $y \in reachable(x)$ {to be parent of $x$}
    $children_S(y) := children_S(y) + x$
    $par_S(x) := y$; $addr_S(x) := addr_S(y) \cdot$ '/'$\cdot name(x)$; $S := S + \{x\}$
  **end if**
```

Lemma 4 (Hierarchical traversal). *For every $x, y \in S$, y is reachable in S from x in typically $O(\log N)$ hops and $O(N)$ in the worst case, where N denotes the number of nodes in S.*

Proof. To be constructed is a route $P(x, y)$ starting at x with the destination EA $addr_S(y)$. The longest common prefix (LCP) in $addr_S(y)$ and $addr_S(x)$, ending in the separator '/', is first computed. Unless it is null, an LCP refers to an ancestor of x in S, so that the tree must be first ascended. Subtracting the prefix from the head of $addr_S(x)$ yields the list of successive nodes leading to the common ancestor. An efficient way to ascend is to count the number of node names in the list, and to forward the packet upwards blindly, decrementing the ascension count until zero. Then, deleting the LCP from the head of $addr_S(y)$ leaves the descension path, of the form $name_1/name_2/\ldots$. At each stage p, the leftmost node name $name_k$ in the path string is looked up in $children_S(p)$: if found, the packet is forwarded to the child where the process repeats for the next node name in the path $name_{(k+1)}$, and so on. The procedure is summarised in Algorithm 2. The performance bounds follow from this tree walking procedure. □

The local lookup efficiency would be similar to that of classic IP and DNS, since only child names need to be matched, only on descent, and their number is given by

Algorithm 2 Interpretive routing

```
Require: packet with destination EA $dest$; fields $rise$ and $index$ in header; one hop
  routing function $nexthop(p) \equiv send(link(x, p))$ at $x$
Ensure: delivery to $dest$
  if $rise = null$ then
    $pfx := LCP(addr_S(x), addr_S(y))$
    $index := 0$; $rise := length(pfx)$ {number of name tokens}
  end if
  if $rise > 0$ then
    $rise := rise - 1$; $index := index + 1$; $nexthop(par_S(x))$
  else
    if $head(dest) \in children_S(x)$ then
      $index := index + 1$; $nexthop(head(dest))$
    end if
  end if
```

the out-degree d, less 1 for the parent. For a large d, an efficient hash table or a search tree could be used at each hop, and require no more than $O(\log d)$ comparisons in the latter case. Since d tends to be rather large especially at the top level domains in the DNS, and furthermore, the ANS is intended to handle routing and signalling in addition to name lookups, there is invariably room for concern about the scalability of the top level nodes of the ANS. The problem would be mostly addressed, in the overall scheme, by the separation of the base network bANS involving the provider nodes and consequently most of the default (canonical) routing, from the secondary dANS trees providing naming specific to applications, geography, etc. Algorithm 2 proves the following corollary.

Corollary 4.1. *Algorithm 1 yields a spanning tree at a cost of $O(N)$ messages.*

Proof. A tree is constructed in Algorithm 1 using at most $O(1)$ messages per node, and using Algorithm 2, every node is reachable from every other node. By definition, this property makes it a spanning tree, having used a total of $O(N)$ messages. □

5.8.2 Lateral routing

Spanning tree routes do not include all available links, however, and are known to be inefficient in this regard. For example, say $addr_S(x) = /A/B/C/x$ and $addr_S(y) =$

$/U/V/y$, and a direct $link(x, y)$ exists; Algorithm 2 will miss this link entirely. More generally, at $x \in S$, non tree links $link(x, p)$ where $p \in S$ cannot be utilised because even knowing p and $link(x, p)$ does not suffice for determining if $link(x, p)$ will usefully lead to y. The only solution is to discover, offline, alternative routes via such direct links, and to augment the route lookup at each node with the discovered information.

Between a pair of nodes x, $y \in S$, $link(x, y)$ will be termed *lateral* if $par_S(x) \neq y$ and $par_S(y) \neq x$. The tree structure comprising only the nonlateral links will be denoted $Tree(S)$, so that $link(x, y)$ is lateral for S iff $link(x, y) \notin Tree(S)$. Then, a route $P(x, y)$ is lateral if for a consecutive pair of nodes $s_i, s_{i+1} \in P$, there is no $link(s_1, s_2) \in Tree(S)$. Let L denote the total set of links between the nodes of S, so that any lateral link $\in L - Tree(S)$. The notion of reachability with lateral links can then be defined recursively as

$$\begin{aligned} Reachable_L^{(0)}(x, y) &= reachable_S(x, y), \\ Reachable_L^{(k)}(x, y) &= \exists\, p : link(x, p) \in L - Tree(S) \wedge Reachable_L^{(k-1)}(p, y), \end{aligned}$$

and correspondingly, a routing function can now be defined as

$$Route_L^{(k)}(x, y) = \langle addr_S(y) \mapsto link(x, p) \rangle, \text{ where } Reachable_L^{(k)}(p, y)$$

so that

$$\begin{aligned} Reachable_L(x, y) &= Reachable_L^{(0)}(x, y) \vee Reachable_L^{(1)}(x, y) \vee \ldots \text{ and} \\ Route_L(x, y) &= \langle addr_S(y) \mapsto link(x, p) \rangle, \text{ where } Reachable_L(p, y). \end{aligned}$$

The power of the ANS tree construction described in Section 5.8.1 is underscored by the following corollary for the reachable subset of x using L, formally defined as $Reach_L(x) = \{y | Reachable_L(x, y)\}$.

Corollary 4.2. $\forall\, x \in S,\ Reach_L(x) = S.$

This merely reaffirms the completeness of ANS as a spanning tree. To actually determine the lateral routes, a route discovery protocol and associated computations are required exactly as done for IP today.

Corollary 4.3. *The routing protocols and algorithms of existing networks suffice for route discovery and computation in S.*

Rationale. In each of the three main classes of routing protocols, distance-vector, link-state and path-vector, the key requirements are the availability of unique node ids, metrics defining link or path costs, and point to point links to carry the routing packets. The construction of S in Section 5.8.1 was based on such links above, and the specification of metrics is largely an administrative function associated with the link setup. Only the problem of unique node ids is new since a manually coordinated address space would defeat the purpose of the present approach and is hence to be avoided. The node EAs themselves, or suitable compact digests thereof, may, however, be used as unique ids for this purpose. $\diamond$

The lateral routes thus discovered would be conceptually represented as a LRT at each node. The availability of tree-walk routes allows two run-time tradeoffs: routing to an ancestor of the destination, identified by a prefix in the destination name, as illustrated by Algorithm 3 below, or to one of its descendents, knowing that the packet can still be appropriately forwarded. Likewise, a packet may be sent up or down the ANS tree on the sending side before being routed laterally across. These tradeoffs can be used to effectively limit the search to a cachable subset of the full set of discovered lateral routes.

Algorithm 3 Lateral routing to an ancestor

```
Require: packet as in Algorithm 2; LRT sorted by longest address first
Ensure: delivery to $y \equiv dest$ by shortest or cheapest path
  if $y \neq par_S(x) \wedge y \notin children_S(x) \wedge \exists$ ancestor $p$ of $y$ with $link(x,p)$ then
    $send(link(x,p))$
  else
    send to $q$ where $q = par_S(x) \vee q \in children_S(x)$ using Algorithm 2
  end if
```

A general problem with caches in both host memory architectures and IP routing is that the penalty for misses is typically high [GC02, DBCP97]. As other traffic would be held up by inline processing of misses, the miss processing is sometimes handed off to another processor in the first case, to obtain what is known as *nonblocking caches* [Sic92]. This cannot guarantee performance, however, as the miss queue can fill up as well. The default tree-walk routes provide yet another way to handle misses, as the missed packets may now be sent off to a parent or even a child of the source node to be properly routed from the latter. On the hypothetical unlimited scale, however, the canonical ANS tree-walk routes would be the only ones possible, since discovery would never be complete!

5.8.3 Best effort flow setup

Routing for the virtual paths must be considered separately from datagram routing, as the label switches in general may not be ANS nodes, and additionally, the routing must be combined with signalling to set up the virtual paths.

The network of label-switching nodes, or "switches", D, will be considered to be connected by point to point datalinks among themselves and with the nodes in S. Each switch is further cohosted with or has a special datalink to at least one node of S, and carries a *virtual path table (VPT).* The VPT entries are numeric tuples $\langle\sigma\,,\pi\rangle$, where σ identifies an outgoing next hop link to another switch or a host, notated $\sigma \to \mathit{link}(,)$ and π is the *virtual path index (VPI)* identifying an onward tuple in the VPT of the next hop switch or an entry into an OS table of application (socket or file) handles if the next hop is an end host. A *virtual path* $v_\pi(x,y)$ then comprises an initial index π and a sequence of nodes $\{x, d_1, \ldots, d_k, y\}$, such that

1. $x, y \in S$, $d_i \in D$, and $\exists$ $\mathit{link}(x, d_1)$, $\mathit{link}(d_k, y)$ and $\mathit{link}(d_i, d_{i+1})$ for $i = 1 \ldots k-1$,

2. $\mathit{vpt}_x[\pi] = \langle\sigma_1\,,\pi_1\rangle$ and for $i = 1 \ldots k-1$, $\mathit{vpt}_{d_i}[\pi_i] = \langle\sigma_{i+1}\,,\pi_{i+1}\rangle$, where $\sigma_1 \to \mathit{link}(x, d_1)$ at x and $\sigma_{i+1} \to \mathit{link}(d_i, d_{i+1})$ at node d_i.

This notion of a VPI is most general, making no distinction between virtual paths and circuits as in ATM. The VPI are just indices into VPTs or the end host OS (file) handle tables, which are managed by the switches (and hosts) themselves. This avoids implied notions of central label managing entities and label distribution protocols

(LDPs), which would then require end to end address coordination, as is the case in the current Internet where the IP address space is layered over MPLS, and is needed for MPLS LDPs. The resulting algorithms can be easily extended, however, to admit label managing entities in small realms or clusters, as in MPLS.

The general path setup problem is thus the routing and construction of an end to end virtual path $v_\pi(x, y)$ given a destination EA. This can be obtained by *end-on* signalling as in ATM [ATM94] if the entire route is known from the source, which is only possible in limited scenarios. The more general solution is guided routing and signalling, developed below.

Lemma 5 (End-on setup). *Assuming there is a means for addressing, a necessary condition for end-on path setup through D is a unique id for each switch $d \in D$.*

The necessity stems from the need to uniquely represent node sequences in route discovery and in the representation of the discovered routes using designated transit lists (DTLs). The number and lengths of the DTLs make their storage requirements more demanding than IP routing tables, which is partly why partitioning is necessary in ATM private network to network interface (PNNI) [ATM94].

The storage performance can be improved using hop-by-hop routing, as in IP, by storing only the next hop address and discarding the tails of the DTLs. If the full DTL for y is available in the LRT at x, end-on signalling is obviously sufficient. Otherwise, the selected LRT entry only points to a next switch. For onward routing, each switch would need a route table as well, but placing route tables at switches presents nothing new over, say, ATM or MPLS. Moreover, the routing would again depend on prior discovery of routes, which is inadequate on the unlimited scale. The ANS then needs to be involved all the way.

Corollary 5.1 (End-on setup with EA). *A necessary condition for end-on virtual path setup using S is to link each switch d as a child of a node s in S.*

Proof. Each switch d then acquires a global EA $addr_S(d) = addr_S(s)/name(d)$, which can be used directly or hashed to form its global id, per Corollary 4.3. The condition is necessary because the only form of a unique id one can have directly from S, which would be available on indefinitely large scales, is the ANS EA. □

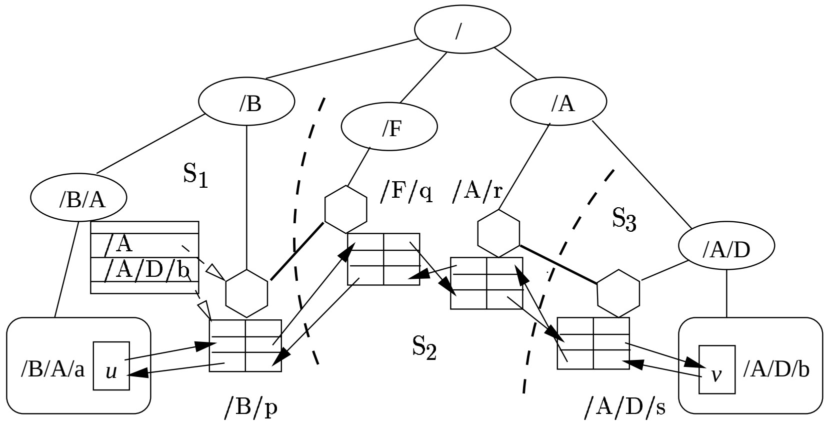

Fig. 5.5: Multi-realm linkage with virtual paths

In Fig. 5.5, an example network is shown comprising an ANS tree represented by the oval nodes, a network of switches shown as hexagons, two hosts as large boxes, switch-to-switch links (thick lines) and the ANS tree links (thin lines). An LRT is shown for the ANS node labelled $/B/A$ on the left, and application processes u and v executing on the two hosts. The scenario concerns process v on host $/A/D/b$, as client, seeking a connection to process u on host $/B/A/a$. For reasons to be soon discussed, the path is best constructed backwards from u towards v. The return EA, $/A/D/b$, dictates an upward hop to node $/B/A$, where the LRT points to switch $/B/p$, and onward link thereon to switch $/F/q$. There, the route must be continued via the ANS tree links to the root / and node $/A$. In cases where switched links exist all the way, the ANS serves merely as a signalling plane.

What is new is the guarantee of routing on scales larger than independent route discovery can support, which would be generally limited to a finite-length numeric address realm around each node or switch. The example network has been redrawn in Fig. 5.6 to illustrate this; the switches are labelled only by their leaf labels for brevity, so that $/A/r$ is simply r. By the same token, the example virtual path may be written $a.p.q.r.s.b$, and would parallel the tree-walk route $a.A.B./.A'.D.b$, where $/A$ has been shortened to A' to avoid ambiguity.

Conceptual radii of network knowledge, representing the routes discovered and represented in the LRTs, have been marked for nodes B and A'. As host b is not within the knowledge radius of node B, there is no way for B to determine the next hop route even to switch p by traditional (layer 3) methods. With the ANS, B is not only able to determine an onward route from p and q, but can also patch the route through the ANS itself by routing to node /, which lies on the ANS tree-walk path.

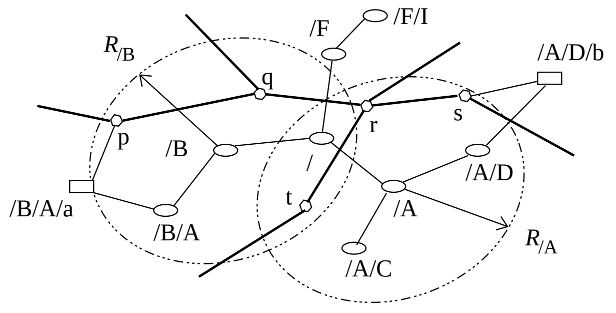

Fig. 5.6: Guided flow routing

Lemma 6 (Guided virtual paths). *A sufficient set of conditions for setting up an end to end virtual path $v_\pi(x, y)$ for $x, y \in S$ are: (a) the connectedness of D from x to y, (b) the routing mechanisms of Sections 5.8.1 and 5.8.2, (c) unique parents in S for each switch in D (Corollary 5.1), (d) the inclusion of switches in route discovery, and (e) proximity of the virtual path route in D to the route in S.*

Conditions (b), (c) and (d) ensure the discovery of routes containing sequences of switched links. For both end-on and broadside-on signalling, the basic LRT entry format must be extended to include switch sequences (DTLs), in general terminating in a non switch node $s' \in S$. The condition for guided routing is that s' be reachable from the current node s via S, that is, that there be a route $P(s, s')$ [or a virtual path $v_\pi(s, s')$ which can serve as P]. The translation rules are then of the form

$$[s_{j-1}|d_{i-1}]\ s_j\ y \mapsto d_i\ \ldots\ d_{i+l}\ s_{j+1}\ y \tag{5.1}$$

where the switch nodes $d_i \in D$ and the final destinations y act as *terminals*. These rules provide a distributed translation from $P(x, y) \equiv \{x, s_1, \ldots, s_k, y\}$ to $v_\pi(x, y) \equiv \{x, d_1, \ldots d_k, y\}$, to be executed along P, and condition (e) means that an appropriate rule of this form can be found at each s_j along the way.

Proof of Lemma 6. This involves generation of the translation rules for each node s_j, selection of a translation rule from s_j to s_{j+1}, and finally, the setting up of the virtual path segment through $d_i \ \ldots \ d_{i+l}$.

The translation rules can be constructed from the *tentative route tree* constructed in Dijkstra's shortest path algorithm [Per99,p.318-319], by adding switch pre-contexts to the implied routes. This leads to Algorithm 4. Although Dijkstra's algorithm is not used in distance and path vector routing protocols, the present ideas are universally applicable as *the route trees can be constructed in reverse from any set of routes.*

Algorithm 4 Synthesis of translation rules

```
Require: at $s_j \in S$: Dijkstra's tentative route tree $TENT(s_j)$
Ensure: compute translation rules for $s_j$
  for all $t \in leaves(TENT(s_j))$ do
    if $t \notin D$ then
      add rule $r : s_j \rightarrow$ path in $TENT(s_j)$ to $t$ till first non switch node
      if first node $u$ on right side of $r$ is a switch then
        for all $w \in D$ such that $\exists\ link(w, t)$ do
          add rule $r(w) : w\ s_j \rightarrow$ same path to $t$
        end for
      end if
    else
      skip $t$
    end if
  end for
```

The incremental complexity in Algorithm 4 for enumerating the neighbours of the switches identified by Dijkstra's algorithm is $O(k)$, where k is the mean in-degree in D. While IP routing would allow gateways within a realm to be fully connected, gateway-to-gateway connections *across* realms, which is where the ANS would be particularly useful, would still be point-to-point and limited to an in-degree much smaller than the total number of gateways in all the participating realms.

It is therefore safe to assume that $k \ll N$ where $N \equiv \#(S + D)$, and further, that k would likely be independent of N, making the incremental cost $O(N^0) \equiv O(1)$ in terms of N. Rule lookup and selection is where most of the complexity lies as an incoming, partially set up virtual path must be matched to an onward path segment, with no guarantee that the onward route will continue all the way to the destination.

Moreover, the penalty for backtracking, which is called *crankback* in ATM, is more severe than in ordinary software practice. It is expedient to assume, for the moment, that this can be solved reasonably well, in order to consider the third step, comprising broadside-on signalling and the stitching of the incoming and onward path segments.

The stitching would be best performed backwards from y to x because at each switch along the intended virtual path v_π, one needs the inbound VPI at the next switch to set up the immediate VPT entry, and this forward VPI is automatically available when traversing S in the reverse direction. The reverse path, from y to x, can then be obtained, if needed, by *end-on* signalling, given in Algorithm 5, which exploits the fact that the forward path already embodies end to end knowledge of the switched route. Given an all-switch route proximal to a canonical path, as stipulated, the forward path setup involves the following events and operations at each switch s_j along the route: (**Stitching algorithm**)

1. Acknowledgement from y arrives at s_j from s_{j+1} bearing a "return EA" (rEA), $addr_S(x)$, identity of the next switch d_{i+l}, and VPI π_{i+l-1} pointing to the onward VPT entry on d_{i+l}. Node s_j picks a rule pointing backwards to y, to identify the list of intermediate switches $d_i \ldots d_{i+l-1}$.

2. Node s_j issues `create-VPT-entry` signalling message to switch d_i, giving it the received VPI π_{i+l-1}, as well as a list of link references $\sigma_i \ldots \sigma_{i+l-1}$ (DTL). Note that the DTL link references would be available at node s_j along with the switch names from $TENT(s_j)$ in Algorithm 4.

   If $l > 1$, switch d_i must execute end-on signalling employing the DTL to set up corresponding VPT entries in $l - 1$ onward switches, terminating on switch d_{i+l-1} with VPT entry $\langle \sigma_{i+l-1}, \pi_{i+l-1} \rangle$, and then its own new VPT entry $\langle \sigma_i, \pi_i \rangle$, where σ_i is returned by the end-on signalling. Otherwise $l = 1$, so that d_i can set up its VPT entry $\langle \sigma_i, \pi_i \rangle$ right away. In either case, d_i returns a new index π_{i-1} pointing to this new entry to s_j.

3. Node s_j propagates the acknowledgement to s_{j-1}, the next node s_{j-1} on the return path (or x if $j = 1$), which was also identified in the rule found in step 1,

but with the switch id changed to d_i and the VPI, to π_{i-1}, in the propagated acknowledgement message. □

The incremental complexity is clearly $O(1)$ so the total cost is proportional to the number of ANS hops, which would be typically $O(\log N)$ and $O(N)$ in the worst case because of the hierarchy (Lemma 4), and thus definitely finite. There is also network latency due to signalling delays unless the gateways are *cohosted* with the nodes of S, which would more efficient for another reason as well, to be explained shortly. The return path construction, given by Algorithm 5, is predictably more efficient.

Algorithm 5 Forward path stitching on return pass (end-on)

```
Require: at switch $d_i$, existing forward VPI $\pi_{i-1}$, partial return path VPI $\bar{\pi}_i$
Ensure: set up forward linkage all the way
  make new VPT entry $\langle \bar{\sigma}_i , \bar{\pi}_i \rangle$ where $\bar{\sigma}_i$ = backward link to requestor
  use $\pi_{i-1}$ to index to forward VPT $\langle \sigma_i , \pi_i \rangle$
  forward request over link $\sigma_i$ with new VPIs $\pi_i$ and $\bar{\pi}_{i+1}$, respectively
```

It is now time to consider the feasibility of efficient rule selection. The problem is illustrated in Fig. 5.7. Say the ANS path follows nodes s_1, s_2 and s_3, and there are many switches d_k within the routing radius of each. The problem at s_1 is to select an onward translation rule for $d_0 s_1 s_2[\ldots y]$, that is, for the left context d_0 and right context $s_2[s_3 \ldots y]$. Without the right context, the problem would be indeterminate, as there would be no destination to route to. A right context of one node s_2 narrows the range of selections to the set $\{d_1, d_2, d_3\}$ that are common to the routing areas of s_1 and s_2. Using two nodes s_2 and s_3 from the right context narrows the choice still further to d_2 alone, from which s_2 would route to one of $\{d_4, d_5, d_6\}$. This representing a narrower "angle of forwarding" $\angle Z s_1 Z'$ than $\angle X s_1 X'$ set by the right context of s_2 alone. The difficulty is that should s_1 chooses d_1 or d_3 as the next hop from d_0, the result could be a longer, or worse, broken path. The existence of an all-switch route all the way, condition (a), is thus necessary, but by no means sufficient.

For instance, assume only the route $d_0.d_2.d_6.d_7$ is satisfiable by the physical links. This would leave a probability $1 - p(d_2|d_0) = p(d_1|d_0) + p(d_3|d_0)$ for the resulting virtual path to break at s_3, requiring either crankback or dynamic patching via the

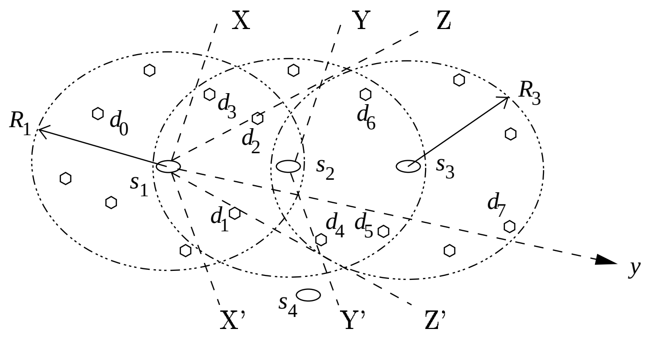

Fig. 5.7: Directional prediction of route

ANS. The probability of good choice, $p(d_2|d_0)$, improves with more right context, as

$$p(d_2|d_0) \leq p(d_2|d_0, s_2) \leq p(d_2|d_0, s_2, s_3) \leq \ldots, \tag{5.2}$$

due to the decreasing angles of forwarding. In the limit of unlimited right context, the forwarding angle reduces to the direction $s_1 \rightarrow y$, and the shortest path computable at s_1. It is therefore desirable to generate as many "forward looking" rules as possible, using Algorithm 4, but we would be still limited to a radius R_i around each node s_i.

It is also clearly impossible to predict the successive nodes beyond s_2 on the initial outbound request from x to y using only treewalk and the LRT rules. However, as the stitching algorithm would be applied on the return pass, the onward node ids s_2, s_3, etc. would be known, and can be captured by tagging acknowledgements with some number of most recently traversed node ids, for augmenting the rules. It is conceivable that all two-node and three-node right context rules can be accommodated, but the best case is still a limited "lookahead". The probability of setting up a virtual path on unlimited scale without patching or crankback is thus generally less than unity.

However, the cost of rule matching and selection is proportional to the number of rules searched. This would be at most k^l, where k is the maximum out-degree of the switches and l is the right context. But k and l are both independent of N, so the cost is $O(N^0) \equiv O(1)$ per guiding node. $\square$ The following result thus holds.

Principle 3 (Networking on unlimited scale). *Canonical routing is practical and efficient even on the unlimited scale.*

CHAPTER 6

ADVANCED TOPICS AND FUTURE WORK

The basic purpose of an architecture to open a gamut of new ways of using it. This chapter discusses some API and OS-like aspects mentioned in Section 1.5:

▷ enhanced *socket* API to reflect the connection-by-name semantics (Section 6.1);

▷ filesystem-like form and security that can be managed without training or skills beyond those ordinarily required for using a host OS (Section 6.2);

▷ automatic recursive aggregation of flows or connections to guarantee scalability of provider networks and ANS gateway relay states (Section 6.3);

▷ a comparable simplified form of IP address/route aggregation that would become available within individual realms using the present approach (Section 6.4);

▷ first-time IP "configuration bootstrap" from a recipe (Section 6.5); and lastly,

▷ efficient handling of large scale mobility and provider changes (Section 6.6).

These ideas are all as yet purely speculative and have not been validated by simulation or prototyping, which would be only possible in a larger project with funded resources. They should accordingly be treated only as visions enabled by the present approach.

6.1 Raising the API – *sockaddr_name*

A straightforward support for ANS addresses would be through a new *address family*, say `AF_NAME`. The corresponding socket address data structure would be defined as in Fig. 6.1, so that a user space virtual memory address can be used for the variable

```
struct sockaddr_name {
    sa_family_t family; // = AF_NAME
    vaddr_t     addrp;  // in user space
    unsigned    length; // in bytes
};
```

Fig. 6.1: Definition of *sockaddr_name*

length ANS name at the system call interface. The corresponding changes necessary in the API may be categorized with *socket* operations as follows:

H-1. Channel parameters and flow control: get/set socket options, push or pop flow control System V STREAMS modules (cf. [Ste90]), etc. These do not involve end point addresses and would therefore remain unaffected.

H-2. Interface access to read the Layer 2 (MAC) address and to configure IP thereon. The addresses are involved only in local administrative role, so there would again be no change in this category. There is no corresponding notion of binding an ANS name to an interface – an ANS node conceptually relates to the host OS and application processes, not interfaces, this being its role as an IDP overlay.

H-3. Layer 3 (IP) address lookup for both local and remote ends, improperly called `getsockname` and `getpeername` in the *socket* API.

The nomenclature is unfortunate, as it conflicts both with Shoch's definitions, according to which a name only says *what* while an address actually says *where*, and with the corresponding distinction in the rest of the Unix OS, for example, between the *name* "eth0" and the USB or PCI bus address that the interface card occupies (e.g. "/dev/usb/4").

We could improve this set, to better support the present views, by introducing analogous functions, `getsockpath` and `getpeerpath` say, that retrieve the high level end point names, returning them as *sockaddr_name*s.

These functions would be far more efficient and reliable than DNS lookups, and "come for free" without additional configuration, as explained in Section 3.4 (D-4, 39). The only catch is with whether the returned remote address belongs to the bANS or to a dANS, which is a higher level than bANS (Fig. 2.2). The latter should be preferred, since it can be dereferenced to the first. The application presumably would have access to the specific dANS, as it could not otherwise participate in that namespace.

H-4. Actual communication: `read/write`, `select/poll` and their variants, signals, which do not use address arguments, and `sendto` and `receivefrom`, which do.

Higher level alternatives could be defined for the last pair, e.g. `sendtopath` and `receivefrompath`, that specifically handle the new *sockaddr_name*, and would be efficient if using the ANS. ANS addresses can be almost as easy or difficult to spoof as IP addresses, so the returned values from these and the `getsockpath` and `getpeerpath` functions must be validated at the application level.

H-5. Connection primitives: `bind`, `connect`, `accept`, `listen`. These Layer 3 *socket* functions have been used without change in all of the preceding examples.

As illustrated in the Chapter 3 examples and more particularly in Section 4.3, the present approach provides an additional user and application level operation, `cbind`, to remote-bind service (ANS address, protocol and port) information. A similar primitive has been used in the prototype to remove such bindings, and implementations may want provide for a lifetime setting option to the `cbind`.

Extensions to create and operate on larger data and program objects at remote ANS locations would amount to treating the ANS as a filesystem, which is discussed in Section 6.2.

A new socket type `SOCK_ANY`, defined by numerical value 0 and a corresponding protocol `NONE`, are also required, as placeholders for returned socket type and protocol number in `getsockpath`, `getpeerpath` and `receivefrompath`. The `connect` system call could be enhanced to accommodate these, as in the code example of Fig. 6.2. The result is a one-step connection setup, which also removes Layer 3 format dependence from user space, so that the same code can now serve for both IPv4 and IPv6.

```
    sockaddr_name addr;
    init(addr);
    int sd = socket(AF_NAME, SOCK_ANY, NONE);
    int fd = connect (sd, &addr, sizeof(addr)); ...
    switch (protocol(addr)) {
        case TCP: /* suitable application behaviour */ ...
    }
```

Fig. 6.2: Use of *sockaddr_name*

This is the proper API approach, as used in modern OSs, for peripheral devices and buses. It may thus be time to depart from the "wooden leg" approach of traditional IP networking in not mapping devices to `/dev`, not defining an elegant standard set of system calls API for network devices, and using dual protocol stacks for IPv6.

6.2 The INternet FileSystem paradigm

Two aspects of the present architecture call for a simple administrative user interface, which, given the high level and the hierarchical form of ANS addressing, appear to ideally suit implementation via the Unix VFS interface (see [Vah96]).

The first is exposure of non canonical routes (i.e. non hierarchical routes in the ANS), particularly including inter-domain IP routes, as proposed in NIRA [*op. cit.*] to end user control. The requisites for this are:

- ▷ an intuitively simple or an already common high level interface,
- ▷ a correspondingly high level representation of the routes to be exposed, and
- ▷ a natural fit between the two.

But for the canonical principle and its OS-like implications, these requisites may seem aesthetic rather than functional. The text form of the ANS permits preferences like to be compiled into the LRTs at any individual node, including an end system host,

```
/b/c :  # probabilities of routing
        via /b/c/switch-pq1a * 0.7 , /b/c/switch-pq1b * 0.3
        else direct             # assumes a direct link in LRT
/a/p/* : direct unless bw > 10 Mbps
        via /x/y unless cost-permin > 0.29 USD
        via /b/router-z102      # unless this not in any route
        via IP                  # assumes same IP realm
        via MAC                 # e.g. same Ethernet
```

Fig. 6.3: Example of routing preferences

a personal hand-held device, etc., and, further, to be personalized for each user on a multi-user host. The simplicity of dealing with high level names instead of numeric IP addresses would greatly encourage use of such preferences whereever supported, as stated in Chapter 1.

The second application concerns network (or subnetwork directory) ownership and access control, based on the following considerations.

I-1. The routing semantics enables security associations to be robustly associated with ANS nodes, thus yielding an easy-to-manage means for this purpose.

For instance, an access constraint could be set on $/s1/h1$ to allow legitimate users from network S_2 to interact with $/s1/h1$, and similarly on $/s1/s2$, to allow access to S_2 only from within S_2:

```
/s1/h2: deny all
        allow /s1/h1 users S1H1 authfile /etc/netauth/s1h1.db
/s1/s2: deny all
        allow /s1/s2 users all
```

Such access control lists (ACLs) are more like those of DFSs and far simpler to use and validate than the ACL in networking today. How they can be actually deployed will be discussed further in Section 6.5.

I-2. VAS enables isolation of portions of a Layer 3 (IP) network by otherwise fully opaque firewalls, frustrating any attempts to bypass bANS authentication via Layer 3 or Layer 2. This is different from simply enabling or disabling protocols, e.g. disabling ICMP (`ping`) response, which is currently done at many public server hosts, because in this case, the ICMP can be selectively enabled for the authenticated remote system administrator.

The analogy is to the Unix *address space isolation* between processes: another process can gain access to the address space of a given process, with suitable privileges, e.g. with the same user or `root` user credentials, and this access is used in most debuggers.

How such an isolation can be rigorously enforced from "bootstrap" up will be discussed in Section 6.5. A similar form of isolation is practised today by reuse of the reserved private address blocks like "net 10" (`10.x.x.x`). but they require manual configuration in each instance. The present approach would be simpler, more general and more scalable.

I-3. The simplicity of the example ACL specification in (I-1) above is due to the specific combination of hierarchical naming with hierarchical routing. An ACL node can readily determine, using the same computation as in routing, whether an ANS address belongs within its "directory tree" or outside of it.

Therefore, lists of address blocks do not need to be specified and categorized to an ANS router in order to configure hierarchical security, which is the simplest and best known security model, used not only in filesystems but also by most corporations, governments, the military, and even primitive societies.

While these possibilities are intriguing, numerous details remain to be worked out for a product, such as how to separate the host root from the network administrative access, distinguish networks from actual filesystems, or handle inadvertent attempts to drag-and-drop documents into router objects from an "explorer" graphical user interface (GUI). Such details are really issues of user interface code, however.

As mentioned in Section 1.5, the INFS kernel extension was implemented on Linux [Gur02b] to examine the usability of VFS encapsulation for ANS and ACL support. INFS specializes the VFS kernel API into a module interface for network namespaces, so that the latter can effectively reuse the OS filesystem name cache and utilize the OS filesystem directory navigation. A DNS resolver module has been implemented to permit usage as below, building on the example of page 17:

```
$ sudo /sbin/mount -o "ns=10.1.0.220" -t dns /inet
$ cd /inet/com/cisco; cd ../ibm/www; w3render < tcp:80
```

Interestingly, as different options can be specified for each mount instance, it is useful to have multiple mount instances. The resolver plug-in does offer specific advantages:

◇ small footprint, due to reuse of VFS name cache and kernel functions;

◇ reduced CPU load, from fewer context switches, than user space resolvers;

◇ direct name-to-connection semantics as with the API of Section 6.1;

◇ and likewise, single socket and user-space stack for both IPv4 and IPv6.

Admittedly, a kernel DNS server would be overkill and impractical – the INFS DNS module is for client hosts, and the idea is more particularly appropriate for a kernelresident ANS node, which would have even less caching requirements and would be able to reuse the OS credentials and authentication mechanisms.

As also mentioned in Section 1.5, there is a known precedent to direct name-to-connection semantics of INFS and *sockaddr_name* in the BSD *portalfs* [SP95]. In both INFS and *portalfs*, the *open(2)* system call takes a pathname string as first (filename) argument and returns an open (client-side) socket descriptor, and all client socket descriptor operations (H-1, H-3 and H-4 in Section 6.1) remain unaltered.

6.3 Recursive, automatic aggregations of flows

It is traditionally believed that stateless routing like IP's is inherently superior to connection-oriented networks in terms of scalability [Hui00a]. In theory, the difference could be substantial, as the switching state volume depends on the local "density" of connections ρ represented by the number of connections through a switch, while IP route tables would be ideally reducible to 3 entries each, as in a binary decision tree. This poses a concern for the present approach as gateway relay states are necessary for inter-realm flows, and for isolating the Layer 3 address spaces of individual realms, in order to avoid the current need for global coordination of these addresses.

In practice, IP networks cannot be aggregated well enough to attain this theoretical limit of 3 entries, and for a good reason: at this limit, there would be almost as many routers as hosts, since each router would be doing very little work, of routing packets by only one bit out of the 32 bits of address! Rather, the situation in the core networks is quite the opposite, with the number of BGP route entries averaging to over 150,000 routes among the large ASs and over 188000 at the largest (Telstra) as of 1 Feb 2005 [Rou]. This is a growth of almost 30,000 routes since 2002 and about 45% since the start of this thesis in 2001.

On the other hand, switching state constitutes *already routed* information, so a substantial compaction should be possible. Using the label stack scheme of MPLS, it is possible to recursively aggregate flows, i.e. combine flows that are themselves aggregates of other flows, permitting a maximum compaction to a size of $O(\log \rho)$ in terms of the original, unaggregated connection density.

A systolic protocol to recursively discover aggregation opportunities and perform the aggregations automatically is briefly described below. As stated in the complete description given in a filed patent application (see Appendix B.3), the protocol can be independently and concurrently initiated by any set of switching nodes, to periodically sweep for aggregable sets of flows, so as to maintain well aggregated average state throughout the network. As will become clear, the very nature of flow aggregates as such ensures that the maximum aggregation would occur in the core networks and the least aggregation would be at the edges. This is an especially convenient property, as a majority of TCP sessions would be too diverse to be usefully aggregated end to end. Current allocation-time aggregation strategies like ARIS [FVWB00a, FVWB00b] are based on the principle that a small set of outer tunnels can be reused for the end user connections. ARIS however depends on the (coordinated) IP address space and the associability of IP routes with statically configured outer tunnels, and the aggregation achieved is thus of a single layer of TCP flows over a single layer of tunnels.

The present protocol does not depend on addressing or an address space, since it only aggregates flows whose routes are already set up, and is inherently more general. It can be combined orthogonally with any system of connection set up mechanisms, including TCP/IP and ARIS, and with the ANS in the present architecture.

Fig. 6.4 shows an example set of unaggregated virtual paths generally going from left to right between the switches $d_0..d_{10}$. Three paths are shown running parallel all the way from d_0 through d_6, and two each from d_0 to d_8, d_7 to d_9 and d_{10} to d_6. These paths may continue beyond these switches, so the protocol should be independently applicable locally over any "compact subgraph" of the network, thus once again using *locality* as the basis for indefinite scalability.

The only further requirement imposed is that it should then be possible to combine the tunnels constructed over a compact region with its peers in adjacent regions with no additional cost of discovery or tunnel construction. To verify that this is possible, consider being given two tunnels t_1 running from d_0 to d_3 and t_2 running from d_3 to d_6. At switch d_3, if t_1 and t_2 encapsulate the same set of component paths, that is, every component path that emerges from t_1 at d_3 gets immediately reencapsulated into t_2, and conversely, every path encapsulated into t_2 comes from t_1, then t_1 may

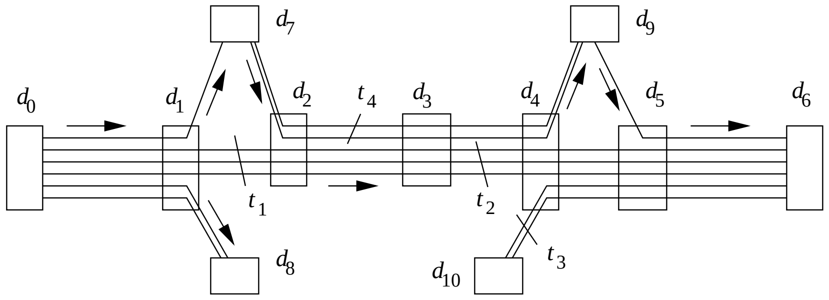

Fig. 6.4: An unaggregated set of paths

be linked to t_2 by simply assigning the outgoing link id and index for t_1 from those of t_2. As the tunnels and the paths they encapsulate represent translated routes, the method closely relates to the notion of *continuations* in functional programming and compilation theory [App92].

There is also scope for *tail-recursion optimisation* (TRO): Consider a tunnel t_3 encapsulating the two paths from d_{10}. Since t_3 is parallel to t_2 all the way to the end of t_2 at d_6, t_3 could be terminated at d_4 using t_2 as its continuation. TRO is only useful when the tunnels terminate at the same switch; if one tunnel continues further, to the right of d_6, a merger with t_2 at d_4 is usable only if the merged tunnel is decapsulated at d_6 to separate the paths, and re-encapsulated, which would be intolerable. One could however bundle a tunnel t_4 comprising the two paths from d_7 to d_9 with the extended tunnel t_1-t_2 via another tunnel t_5 from d_2 to d_4. The net result is shown in Fig. 6.5 with t_5 being depicted by shading. Although a very small number of paths was considered for simplicity, the reduction of switch-table entries to $O(\log \rho)$ is clearly discernible. The protocol is characterized by the following lemma.

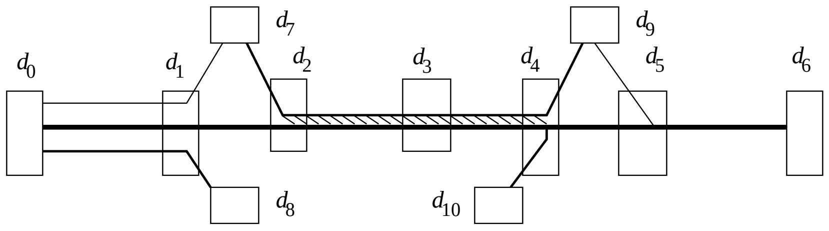

Fig. 6.5: The same paths after aggressive aggregations

Lemma 7 (Lossless flow autoaggregation). *The following procedure is lossless.*

Discovery: Switch d_0 initiates the whole process by examining its virtual path tables and identifying a list of paths all going to the same next switch d_1. It then issues a *discovery* packet bearing the list of indices (labels) corresponding to these paths to the next switch d_1, which looks up the listed indices to obtain the corresponding onward indices for propagating the discovery packet further. As the paths split into two groups, d_1 correspondingly splits the list, and then has the option to propagate the discovery packet along the largest subgroup only, going onwards to d_2, or along each of the subgroups, thereby also including d_7. The discovery packet includes a hop count which increments to 6 by the time the packet arrives at d_6.

The last switch d_6 would be identified by some criteria, the list size dropping below a threshold which may be locally configured, the hop count exceeding a threshold, or by policy for lists arriving from certain peers.

Association: Switch d_6 allocates a new path table entry for the tunnel and *reflects* the discovery packet back towards the initiator d_0 along with an extra index (label) corresponding to the new tunnel. At each backward hop, the receiving switch does a reverse-lookup for each of the listed path indices to identify the corresponding incoming indices, and creates a new entry for the tunnel with the received tunnel index. The reverse propagation stops when the hop count drops to 0 (at d_0), identifying the outbound paths associated by the route.

Aggregation: The original initiator d_0 then marks each of the identified paths as redirected to the new tunnel entry, and issues an aggregation command with the identified list of paths and the new tunnel path index. This is propagated similarly all the way to d_6, identified by decrementing a copy of the hop count obtained in the discovery pass.

Finalise: Last switch d_6 returns a *finalisation* packet as acknowledgement. At each switch *en route*, the entries for the component paths are discarded.

Rationale. If the switches are not otherwise overwhelmed, a packet can get dropped if the onward path table entry corresponding to its index happens to be missing when it arrives. The tunnel already exists all the way after the association signal, and the decapsulation at its tail end is implicitly arranged by the last switch d_6 before the association signal is even issued. Thus, when d_0 redirects the identified paths, the only window for losing packets would be during the setting up of the redirection at d_0, which can be made robust by appropriate software locks or semaphores. $\diamond$

Lemma 8 (Fast autoaggregation). *Autoaggregation can be performed in 2 passes with limited packet loss.*

Proof. Consider discarding the entries for the component paths immediately with the association signal, so that the tunnel setup and aggregation are both completed when d_0 redirects the component paths, obviating the last two passes.

If n is the number of hops, and if the mean hop delay Δt is much greater than the processing delays at each switch, the component paths would be dismantled over a period $(n-1)\Delta t$. However, as the dismantling occurs from the tail end, the packets arriving with a just dismantled path index at a switch m hops downstream from d_0 would have arrived at d_0 at a time $m\Delta t$ earlier. The number of packets lost would be $L \approx n(n-1)r\Delta t/2$, where r is the mean arrival rate of packets at d_0. Since $L = O(n^2)$, it would seem impossible to limit the packet loss in an indefinitely large network. The trick is to configure a limit on n given that the resulting tunnel segments can be efficiently concatenated, as explained for t_1 and t_2 in Fig. 6.4. $\square$

Lemma 9 (One-pass deaggregation). *Deaggregation can be done losslessly in one end to end pass, but using same number of messages as two passes.*

Proof. Discovery is unnecessary as the route is already marked by the tunnel, and the components are identified by the redirection entries at the first switch of the tunnel. All that is required is to create, at each of the successive switches, new path table entries for the tunnelled components and to return the list of their indices to the preceding switch before propagating the command to the next switch. The rest of the reasoning is trivial. $\square$

Lemma 10 (Efficient recovery). *Tunneling can simplify healing.*

Rationale. The idea is to store the tunnel end point EA at the head of the tunnel, so that the tunnel path can be reconstructed when necessary. Correspondingly, entries of the original broken path may be timed out at the respective switches, and timing out used avoid teardown signalling altogether. The result could be a more efficient cache-like, best-effort virtual path network. This would be more efficient than healing each component path separately, and trades off storage for the tunnel EAs. ◇

6.4 Route-directed address aggregation

It might be expected that an analogous form of aggregation be also possible for the traditional Layer 3 (IP) addresses, on grounds that the elimination of global address space coordination in the present approach is in some senses complementary to the already-routed nature of switching states. We should not need to be concerned about the latency of address change updates in the DNS or BGP, nor about the optimality of Layer 3 address assignments with respect to external route tables. The only remaining constraints on renumbering would be

- ◇ smooth transition of active connections, like those of TCP and SCTP,
- ◇ consistency of the addresses with the node names at the ANS nodes within the realm and at the ANS gateways for incoming packets,
- ◇ and consistency of Layer 3 routing within the realm.

Several strategies for smoothly maintaining or resuming connections across address changes are known in the literature, and are uninteresting to the context.

The address and routing consistency requirements are considerably less than in the current Internet, however, when one identifies realms with the individual ASs or end user networks of today, so there is considerably more leeway with respect to address space usage and renumbering. Since the (Layer 3) addresses are not relevant outside of the immediate realm boundary, an optimal renumbering would be concerned with traffic patterns only inside of the realm. This optimal allocation would be only known after the routes have been discovered, but the discovery *per se* depends only on unique node ids, not necessarily routable addresses. This presents yet another opportunity to reuse routing information in reverse.

The procedure *per se* is simple: one may pick any node r as the addressing root and allocate addresses recursively for its routing subtree $rt(r)$ constructed from Dijkstra's algorithm or from BGP data, as explained in Section 5.8.3. The result, Algorithm 6, is very similar to Huffman numbering in coding theory, the difference being that the hierarchy is of routes instead of probabilities, and is not binary. Actual renumbering schemes that have been employed in the past involve more complex considerations of traffic distributions.

Algorithm 6 Efficient address allocation $addrsp(S, r, t, a, m)$

```
Require: network $S$; root node $r \in S$; route tree $t \equiv rt(r)$; prefix $a$; mask $m$
Ensure: efficient address allocation for nodes in $t$
  let $sp = \phi$ {set of address mappings to be returned}
  let $k = \#children_t(r)$; {children of $r$ defined by the routing tree $t$}
  let $kb$ = smallest $(\{2^k|2^k \geq x\})$; $km = kb - 1$; $ka = 0$
  for all $s \in children_t(r)$ do
    $sp = sp + \langle s\,, (a.ka)\rangle + addrsp(S, s, a.ka, m.km)$; $ka = ka + 1$
  end for
  return $sp$
```

The contribution of the present approach would be not so much in simplifying the allocation or address changeover, but in permitting greater flexibility in the choice of address range, for example, the entire of IPv4 space, or a randomly chosen block of IPv6 addresses, since the realm Layer 3 addresses would not be visible outside of it. Use of Layer 3 addresses within applications as peer end point identifiers is as such inadvisable today because of firewall transparency issues, and stronger arguments have been made in Chapter 1 for using application level identifiers.

The above procedure may be also applied within regions of a single Layer 3 (or Layer 2) network, when isolated by a firewall, for instance, using VAS as suggested in item I-2, on page 103. In such a region S, only a gateway of S can be chosen as root or else multiple route entries will be needed for S everywhere else in the realm. Selection among the gateway nodes must be then based on minimising the total consumption of addresses, as summarised in Algorithm 7 below.

One simple way to ensure smooth changeover without disrupting active connections would be to simply retain the old set of addresses, and route table entries, for as long

Algorithm 7 Address autoaggregation

```
Require: network $S$; gateway nodes $G \subseteq S$; prefix $a$; mask $m$
Ensure: most efficient address allocation and compaction of route tables
  for all $g \in G$ do
    compute $t = rt(g)$
    compute efficient address allocation $addrsp(S, g, t, a, m)$ (Algorithm 6)
  end for
  root $r$ = select $g \in G$ with smallest $addrsp(S, g, t, a, m)$
  apply the computed allocation $addrsp(S, r, t, a, m)$ to $S$
```

as the prior connections exist, after updating the associated ANS or DNS with the new addresses to ensure that all new connections use the new ones. The availability in the present approach of the entire IPv4 address space at one's disposal makes this solution generally available and easy to apply.

Lemma 11 (Efficient subnet changeover). *The smallest set of addresses that need temporary duplicate route entries corresponds to the largest set of hosts attached to the same router.*

Rationale. One may start by applying the new allocation with the lowest level router supporting the largest number of hosts. This entails at most one additional prefix in all higher level routers. The specified minimum can be achieved by applying the new addresses to the hosts of at most one router at a time. $\diamond$

One may prefer to migrate active TCP connections to the new addresses anyway. Tools for such purposes exist, e.g. as in a namespace mobility scheme [SB00]. The ideas of this section are thus for illustration only, and by no means special.

6.5 IP bootstrap discovery and configuration

A further illustration of the OS and filesystem paradigms is to be able to *bootstrap* (or *format*) even the first-time IP configuration using a tailored recipe on a central host that specifies the desired INFS directory and permissions structure. This may not be widely needed as most networks already have IP configured. However, Layer 3 renumbering, for routing efficiency, can be simple and efficient as suggested in Section 6.4, and this raises the question of whether the first time configuration for a new Layer

3 network would also be correspondingly possible. Other motivations would be for *ad hoc* networking using the present canonical approach and for quick recovery.

For this, the ANS nodes and routers would be assumed to start in unconfigured initial states, and the Layer 2 infrastructure needs to be discovered. Such discovery is performed today in many OSSs, but it generally depends on the availability of IP. The challenge for "IP bootstrap" (or "IP format") is to discover using only Layer 2 drivers and addressing, subsuming the bridge spanning tree protocol in the process, among other things, and to correctly identify and match the nodes with their specified roles in the recipe. As a result,

- ◇ the discovery itself has to be *everywhere local*: there can be no communication between the discovery frontier and the controlling host;
- ◇ and the frontier must assign (IP) addresses right away so that communication can be set up using Layer 3 for implementing the recipe configuration.

Principle 4 (Bootstrap IP configuration). *First time discovery of a network and its IP configuration can be accomplished using $O(N)$ messages, where N denotes the number of nodes to be discovered and configured according to a centrally located recipe, provided the subset of the nodes identified in the recipe bear identifying information.*

Algorithm.

The key observation is that at each node in a tree topology, which generally describes an ATM-like switched network, each neighbouring node can be identified *locally* by the link interface leading to it. Then, in a first discovery pass starting from the recipe host, successive rings of new nodes can be thus identified. Loops can be avoided by marking each newly discovered node with a "discovery generation" number assigned by the recipe host and a pseudo-random id generated by its preceding host, both numbers arriving in the discovery packet. This initial discovery terminates automatically at the leaf nodes (hosts) and must be terminated at the gateway routers to the public network by some means, to be shortly examined.

Then, in a return pass starting at the most recently discovered nodes (hosts and gateway routers), the ids are returned to the preceding nodes, thus providing each of the latter with a count of its descendents in the discovery tree traced in the first pass, which has the recipe host as root. When all of the root's neighbours have returned their respective counts, a simple, first-time IP configuration can be applied starting at the recipe host, and assigning, at each node, successive large enough address blocks, determined by the previously acquired counts, to each of its immediate descendents, and reflecting these assignments in its own configuration as an IP router.

Once this pass completes and has been acknowledged all the way back to the root, the ACLs specified in the recipe can now be associated with the assigned IP addresses and downloaded to the respective router nodes for application.

Ethernet and wireless networks present a complication that at each node, a single physical interface leads to multiple neighbours that cannot be distinguished *a priori*. One solution is adapt the "zeroconf" self-numbering technique by letting each such neighbour assign itself a trial IP address based on its MAC, broadcast its selection to its neighbours, and iteratively pick from a related sequence of addresses if collision is detected. The process rarely goes beyond the first choice, and as each node requires only $O(1)$ iterations before settling down to one, the overall process remains $O(N)$ in the number of messages sent or received. The other complication is the termination of the discovery required at the gateway routers, but this can be safely automated if each such router is set up to distinguish between internal and external interfaces. $\diamond$

The bootstrap configuration cannot be ideal for Internet routing. It is preferrable to designate a principal gateway router as the root of the interior address hierarchy if much of the IP traffic is expected to be to and from the Internet, as considered for Algorithm 7 in Section 6.4. It would be adequate as an initial IP configuration, however, using which a more suitable renumbering can be subsequently applied. In the present architecture, it principally serves to demonstrate, in support of the canonical construction. that even the local realm configuration can be usefully eliminated.

6.6 Super-scalable multihoming and mobility

It was broadly stated in Chapter 1, specifically on page 16, that a subscription-based update mechanism would generally suffice for efficient handling of server mobility and provider changes, and the associated problem of multihoming. The current Internet problem of multihoming actually comprises two issues:

- ▷ BGP route table explosion from the proliferation of small address blocks that result from users subscribing to multiple providers and advertising all of their addresses to maintain reachability during provider failures;
- ▷ problem of renumbering user networks to switch providers and of the latency of route discovery and DNS updates to reflect the changed numbers (addresses).

Carrier networks currently "solve" this by blocking advertisements of small address blocks, thereby denying a workaround for provider failures to the smaller users. While the first problem would be alleviated, in both NIRA and the present architecture, by the default canonical routing, the second problem would at first appear to be worsened in the present scheme because provider nodes need to be visible in the ANS.

The solution, introduced on page 16, is that of derivative ANS (dANS) trees devoid of provider references; these dANS would suffice for provider-independent addressing, but would need to be dereferenced to the base ANS (bANS) containing the provider nodes for actual routing. The dANS nodes must carry routing information similar to LRTs, therefore, and need to be updated both whenever member leaf nodes migrate and also whenever any of their ancestors within the bANS is repositioned. While the individual leaf node migration can be well handled by actively updating each of the relevant dANSs, this being very quick and efficient in the present approach, it is not obvious that higher level provider changes can be scalably and efficiently handled. Both the basic scheme of routing with the dANS and an efficient and scalable method of handling large scale network or provider relocations are treated below.

6.6.1 Use of a derivative ANS

Routing between any pair of end nodes, which are not in the same subscriber network, has to follow links to their respective providers, and thereon to a common "cross-over" router, and subsequently the same links back to the end nodes, so the base network is in effect a "realization layer". Fig. 6.6 illustrates the role of provider nodes in the scenario of Fig. 3.5 (page 40), wherein the IP links between gateways g_A and g_B are routed through a provider network E via E's boundary routers g_{E1} and g_{E2} and a segment interior to E. Like Fig. 3.5, Fig. 6.6 concerns the Layer 3 "data plane" rather than signalling. The following lemma was implicitly used in Sections 5.4, 5.5 and 5.8.

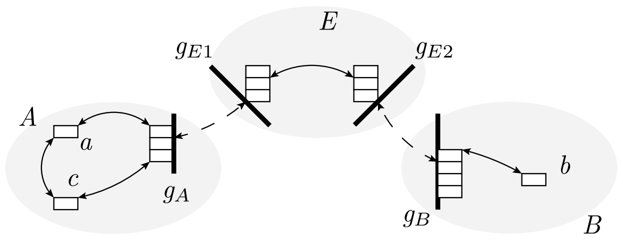

Fig. 6.6: Involvement of the provider network

Lemma 12 (Interior routing). *Every intermediate realm needs a bANS node.*

Rationale. Together with g_A and g_B, node e would need to set up the incremental route segments $g_A \leftrightarrow g_{E1}$, $g_{E1} \leftrightarrow g_{E2}$ and $g_{E2} \leftrightarrow g_B$. The routing over the outer links $g_{E2} \leftrightarrow g_B$ and $g_A \leftrightarrow g_{E1}$ would be determined from the application service registration and the client connection request, respectively, as described in Chapter 3. The middle link $g_{E1} \leftrightarrow g_{E2}$ requires Layer 3 addresses $addr_E(g_{E1})$ and $addr_E(g_{E2})$ which are private to E. Node e has to be within E in order to access $addr_E(g_{E1})$ and $addr_E(g_{E2})$, and this would hold for any number of intermediate realms. $\diamond$

6.6.2 Large scale mobility by subscription

The solution of secondary name trees as mentioned in the preceding sections is partly DNS-inspired, in that the problem of explicit provider dependence is eliminated by use of an application level name space that dereferences to actual routable addresses. The difference is that in the present approach, the routable addresses are themselves name-like, ultimately belonging to the bANS, and that the dANS would be themselves routable as well in a logical sense, permitting other name trees to dereference to them, so that multiple layers of indirection are trivially possible. The canonical principles permit simpler treatment of mobility and provider change or migration as follows.

The need for updating dANS nodes arises as follows. Consider first the case of a node a needing to relocate from its initial position given by its base (bANS) address $addr_1(a) \equiv$ `/e1/e3/e4/gA/a` to `/e1/e2/e5/a`, a common example being that of a notebook PC plugged into a corporate network during day and connected directly to the Internet when at home. The set of nodes affected comprises all members of all dANSs in which node a is a participant. This set would be usually much less than the full base network, which would correspond to the entire Internet. Each of these peer nodes could reach a via the bANS, but the translations from $addr_{S'}(a')$, where S' is a dANS and a' is the identity of a in S', to its new location, $addr_2(a) \equiv$ `/e1/e2/e5/a`, cannot be known until an explicit update or rediscovery.

Node a can thus only resume its identity and function in each dANS S', S'', etc. after it finishes updating its corresponding registrations a', a'', etc. therein, so that messages or connections from its clients or peers can be properly forwarded to a. All acknowledgements would be routed to the new location instantly in any case.

Alternatives would be to allow the old address $addr_1(a)$ to fail when forwarding is attempted, or, taking from the postal system, to arrange for the returning of an updated address, and to thus avoid re-registration traffic. The latter would be simpler, but cannot scale to highly dynamic environments and would leave applications and users with no efficient means to reconnect with relocated nodes. Re-registrations are thus generally needed and generally suffice for the mobility of leaf nodes.

A naïve way to handle address changes originating within the provider hierarchy would be for each leaf node a to subscribe for notifications of such changes by its parent. However, this would produce an exponentially increasing set of notification packets propagating down the hierarchy at each such change, and the notifications would then cause a spate of outbound update messages from each leaf node. Worse, as many leaf nodes would be likely participating in the same dANSs, the same provider address change message would arrive at each dANS multiple times. It is shown next that the dANS maintenance would in fact be remarkably efficient.

Principle 5 (Efficiency of derivative maintenance). *Any number n of dANSs can be subscribed and notified of provider changes, with storage and update traffic of at most $O(n)$ complexity, and without global coordination.*

Rationale. In terms of the preceding examples, the problem of the naïve approach can be understood by considering a relocation of g_A from `/e1/e3/e4/gA` $\equiv addr_1(g_A)$

to `/e1/e5/gA` $\equiv addr_2(g_A)$. This would require a and its siblings belonging to g_A to re-register in each of their respective dANSs. Moving the provider node e_4 from `/e1/e3/e4` to `/e1/e5/e6/e4` would impact all descendents of g_A's siblings as well.

Consolidation of notifications and updates seems to be efficiently achievable using set union operations as follows. The relocated node, g_A or e_4 in the above examples, would need to carry a list of dANSs to be updated upon relocation. It is not possible for g_A to get the list reliably by monitoring member traffic, as the real destinations can be easily concealed even without encryption, even if the subscriber credentials were not required for the actual updates. The only reliable way is instead to have each leaf node subscribe to its parent g_A, along with its list of dANSs to be notified and the requisite credentials. As these lists will have common entries, the parent needs to compute a union of the dANS references. It may then subscribe to its own parent in the bANS, giving the union list as its list of dANSs to update.

For this to work, it is necessary to also have a means for uniquely identifying and addressing each dANS, as well as a trusted broadcast provision in the latter so that only one of its nodes needs to be notified of the address changes.

Both requirements can be met, *without reintroducing global coordination*, by using the base (bANS) addresses of the root nodes of the dANSs and a hash table to obtain their union. For example, if $a \sim$ `/e1/e3/e4/gA/a`, $b \sim$ `/e1/e3/e4/gA/b` and $c \sim$ `/e1/e3/c` were involved in exactly one dANS S', and subscribed to their respective parents for change notifications, their update requirements at e_3 would be exactly one entry, $root(S')$. This hash table and dANS broadcast ensure linear complexity.

One limitation is that bANS address changes affecting any dANS roots cannot be automatically notified, which means that these roots have to be themselves responsible for updating their remaining members. This means that a provider node should not update any dANSs whose roots are among its descendents. An elegant way to ensure this is for the lower nodes not quote any dANSs of which they (the lower nodes) are themselves roots, and to reject, at their parents, any inadvertent subscriptions of this kind. This would further decrease the storage required at the provider nodes. $\diamond$

CHAPTER 7

CONCLUSION

The net result is a fundamental theory and architecture for networking, i.e. for both addressing and routing, with the fundamental advantages of being coordination-free, protocol-agnostic and scale-invariant.

Numerous alternative proposals concerning network address or routing, too many to be faithfully cited, were examined during the course of this work, if for no other reason than as sanity check and possible mid-course correction over the 5 years that this work has taken. None come close to matching these properties even in theory. The most significant of other proposals, notably TRIAD, IPNL, FARA and NIRA, as well as partial schemes like DHT have been discussed and compared with in detail, principally in Section 1.3. None of those alternatives appears to have been actually deployed on a significantly larger scale than the present scheme, the only alternative to IPv4 currently deployed on a substantial scale being IPv6, hence the only technical measures of comparison would be the theory and the depth and extent of analysis.

TRIAD, for example, advocates name-based routing, but stops short of providing an efficient absolute address space mechanism using names. This was to be expected given that it stands for "active directories", which presupposes *a priori* existence of a lower level global address space that would by definition be responsible for the actual routing (cf. [Sho78]). IPNL fundamentally constrains itself by NAT, as explained in Section 3.1. FARA and NIRA fundamentally propose a second coordinated number space, essentially doubling the coordination and human involvement required, which seriously questions the simplicity of these schemes and their expectation of greater flexibility and dynamism to result in the Internet. NIRA does share the idea of default hierarchical routing, which would avoid the current dependence on the completeness and reliability of BGP discovery and the scalability of BGP route tables, that none of the other schemes measure up to. It thus lends support to the present approach.

The present thesis not only introduces a more fundamental theory of networking but also treats or demonstrates, among other aspects,

- substantial simplifications over current practice, such as not having to register names and addresses separately and not needing a global registry and agencies to coordinate the addresses;
- IP-like first-connect times subsuming name resolution, which is unexpected from the experience and measurements with the DNS [JSBM01], and other naming, name space or name-based routing schemes including CAN [RFH$^+$01] and INS [AWSBL00] – the performance becomes understandable on noting that the first-time connections comprise IP-like routing by name paths, and signalling and channel setup over *already* existing (TCP) links, and thus similar to that of `ssh` port forwarding;
- instant response to topology changes routinely demonstrated by the prototype, since no discovery is necessary for the canonical routes to become active;
- and large scale self-management including mobility and "self-renumbering", as discussed in Section 6.6.

All of these features are consistent with, and expected of, a fundamental theory, and will hopefully suffice to motivate further study and deployments of this approach.

REFERENCES

[ABKM01] D G Andersen, H Balakrishnan, M F Kaashoek, and R Morris. Resilient Overlay Networks. In *SOSP*, 2001.

[App92] A W Appel. *Compiling with Continuations*. Cambridge, 1992.

[ATM94] The ATM Forum. *Private Network-Node Interface Specification, af-pnni-0026.000*, 1994.

[ATM02] The ATM Forum. *ATM User-Network Interface (UNI) Signalling Specification version 4.1, af-sig-0061.002*, 2002.

[AWSBL00] W Adjie-Winoto, E Schwartz, H Balakrishnan, and J Lilley. The Design and Implementation of an Intentional Naming System. In *SOSP'00*, 2000.

[BLR$^+$04] H Balakrishnan, K Lakshminarayanan, S Ratnasamy, S Shenker, I Stoica, and M Walfish. A Layered Naming Architecture for the Internet. In *Sigcomm 2004*, 2004.

[Car00] B Carpenter. Internet Transparency. RFC 2775, Feb 2000.

[CBFP03] D Clark, R Braden, A Falk, and V Pingali. FARA: Reorganizing the Addressing Architecture. In *ACM FDNA 2003 Workshop*, Aug 2003. See http://www.isi.edu/newarch.

[CC78] D D Clark and D Cohen. A Proposal for Addressing and Routing in the Internet. IEN 46, Jun 1978.

[CCR97] B Carpenter, J Crowcroft, and Y Rekhter. IPv4 Address Behaviour Today. RFC 2101, Feb 1997.

[CCS96] I Castineyra, J N Chiappa, and M Steenstrup. The Nimrod Routing Architecture. RFC 1992, Aug 1996.

[CG00] D Cheriton and M Gritter. TRIAD: A Scalable Deployable NAT-based Internet Architecture. Stanford Computer Science Technical Report, Jan 2000.

[Che99] D R Cheriton. Translating Relaying Internetwork Architecture integrating Active Directories (TRIAD), 1999. See `http://www-dsg.stanford.edu/slides/triad-wrap-netseminar`.

[Coh78a] D Cohen. On Names, Addresses and Routings. IEN 23, Jan 1978.

[Coh78b] D Cohen. On Names, Addresses and Routings (II). IEN 31, Apr 1978.

[CPRW03] D D Clark, C Partridge, J C Ramming, and J T Wroclawski. A Knowledge Plane for the Internet. In *SIGCOMM*, 2003.

[D$^+$97] R Droms et al. Dynamic Host Configuration Protocol. RFC 2131, Mar 1997.

[DBCP97] M Degermark, A Brodnik, S Carlsson, and S Pink. Small forwarding tables for fast routing lookups. In *SIGCOMM*, 1997.

[Dee00] S Deering. Re IPv6: Past mistakes repeated?, Apr 2000. IETF mail archive.

[(Ed04] M Brunner (Ed.). Requirements for Signaling Protocols, April 2004.

[Ege94] K Egevang. The IP Network Address Translator (NAT). RFC 1631, May 1994.

[FG01] P Francis and R Gummadi. IPNL: A NAT-Extended Internet Architecture. In *SIGCOMM*, 2001.

[FGM$^+$99] R Fielding, J Gettys, J Mogul, M Frystyk, L Masinter, P Leach, and T Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Jun 1999.

[For03] B Ford. Unmanaged Internet Protocol. In *HotNets-II*, 2003.

[Fra94] P Francis. Pip Near-term Architecture. RFC 1621, May 1994.

[FVWB00a] N Feldman, A Viswanathan, R Woundy, and R Boivie. Aggregation of data flows on switched network paths. US Patent 6,069,889, 2000.

[FVWB00b] N Feldman, A Viswanathan, R Woundy, and R Boivie. Mapping of routing traffic to switching networks. US Patent 6,055,561, 2000.

[G^+94] A Geist et al. *PVM: Parallel Virtual Machine...* MIT Press, 1994.

[GB01] M Gainor and S Bradner. Firewall Enhancement Protocol. RFC 3093, Apr 2001.

[GC01] M Gritter and D Cheriton. An Architecture for Content Routing in the Internet. In *USITS*, 2001.

[GC02] K Gopalan and T Chiueh. Optimizations for Network Processor Cache. citeseer.nj.nec.com, 2002.

[Gur00] V Guruprasad. Who Needs Addresses? In *ECUMN*, Oct 2000.

[Gur01a] V Guruprasad. Above and Beyond the Internet. In *NY Metro Area Networking Workshop*, Mar 2001.

[Gur01b] V Guruprasad. OSIx: Operating System Internet. In *INET'2001 poster*, Jun 2001.

[Gur02a] V Guruprasad. A self-managing addressing, naming and routing service for appliances. *Fourth Intl. Workshop on Networked Appliances*, Jan 2002.

[Gur02b] V Guruprasad. The INternet FileSystem (INFS), 2002. http://infs.sourceforge.net.

[Gur03] V Guruprasad. Autonomic Addressing and Internetworking on Unlimited Scale. *Annales des Telecommunication*, 58(7-8), Jul-Aug 2003.

[Hai00] T Hain. Architectural Implications of NAT. RFC 2993, Nov 2000.

[HJ98] M Handley and V Jacobson. SDP: Session Description Protocol. RFC 2327, Apr 1998.

[HS01] M Holdrege and P Srisuresh. Protocol Complications with the IP Network Address Translator. RFC 3027, Jan 2001.

[HSSR99] M Handley, H Schulzrinne, E Schooler, and J Rosenberg. SIP: Session Initiation Protocol. RFC 2543, Mar 1999.

[Hui00a] C Huitema. *Routing in the Internet*. Prentice-Hall, 2000.

[Hui00b] C Huitema. *Routing in the Internet*, chapter 6. Prentice-Hall, 2000.

[JSBM01] J Jung, E Sit, H Balakrishnan, and R Morris. DNS Performance and the Effect of Caching. In *ACM SIGCOMM Internet Meas. Workshop*, 2001.

[KRS02] R Krishnan, R Ramanathan, and M Streenstrup. Optimization Algorithms for Large Self-Structuring Networks. In *IEEE INFOCOM*, 2002.

[McN99] D McNab. BSD Portals for Linux 2.0. NASA Tech Report NAS-99-008, Oct 1999.

[Moc87] P Mockapetris. Domain Names – Concepts and Facilities. RFC 1034, Nov 1987.

[Mos01] R Moskowitz. draft-moskowitz-hip-arch-05.txt, Nov 2001.

[MPI97] The MPI Forum, `http://www-unix.mcs.anl.gov/mpi`. *The Message Passing Interface (MPI) standard*, 1997.

[MR04] N Modadugu and E Rescorla. The Design and Implementation of Datagram TLS. In *Proc. NDSS*, 2004.

[NLM96] P Newman, T Lyon, and G Minschall. Flow Labelled IP: A connectionless approach to ATM. In *IEEE INFOCOM*, 1996.

[NZS01] T S E Ng, H Zhang, and I Stoica. A waypoint service approach to connect heterogeneous Internet address spaces. *Usenix Technical Conf*, Jun 2001.

[OY02] L Ong and J Yoakum. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286, May 2002.

[Per99] R Perlman. *Interconnections*. Addison-Wesley, 1999.

[PHG^{+}96] J Porter, A Hopper, D Gilmurray, O Mason, J Naylon, and A Jones. The ORL Radio ATM System. TR 96.5, http://www.uk.research.att.com, Jan 1996.

[Pos81] J Postel. Internet Control Message Protocol, September 1981.

[RCCD02] J Rajahalme, A Conta, B Carpenter, and S Deering. draft-ietf-ipv6-flow-label-02.txt, Jun 2002.

[Rea03] M Rearden. Rush to VoIP turns to slow going. *CNET News.com,* `http://news.com.com/2100-7352-5133200.html`, Dec 2003.

[RFH^{+}01] S Ratnasamy, P Francis, M Handley, R Carp, and S Shenker. A Scalable Content-Addressible Network. In *ACM SIGCOMM*, 2001.

[RL95] Y Rekhter and T Li. A Border Gateway Protocol 4. RFC 1771, Mar 1995.

[Rou] `http://bgp.potaroo.net/cidr/`.

[RS04] V Ramasubramanian and E G Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. In *Sigcomm 2004*, 2004.

[RVC01] E Rosen, A Viswanathan, and R Callon. Multiprotocol Label Switching Architecture. RFC 3031, Jan 2001.

[SB00] A Snoeren and H Balakrishnan. An End-to-End Approach to Host Mobility. In *6th ACM MOBICOM*, Aug 2000.

[SC03] H Schulzrinne and S Casner. RTP Profile for Audio and Video Conferences with Minimal Control, July 2003.

[SCFJ03] H Schulzrinne, S Casner, R Frederick, and V Jacobson. RTP: A Transport Protocol for Real-Time Applications, July 2003.

[Sho78] J S Shoch. A Note on Inter-Networking Naming, Addressing and Routing. IEN 19, Jan 1978.

[Sic92] J E Sicolo. A multiported nonblocking cache for a superscalar uniprocessor. Master's thesis, UIUC, 1992.

[SMS$^+$00] K Stewart, K Morneault, M Schwarzbauer, T Taylor, I Rytina, M Kalla, L Zhang, and V Paxon. Stream Control Transmission Protocol. RFC 2960, Oct 2000.

[SP95] W R Stevens and J S Pendry. Portals in 4.4BSD. In *USENIX Tech Conf*, Jan 1995.

[SRC84] J H Saltzer, D P Reed, and D D Clark. End-to-end arguments in system design. *ACM Trans Comp Sys*, 2:277–288, Nov 1984.

[SRL98] H Schulzrinne, A Rao, and R Lanphier. Real Time Streaming Protocol. RFC 2326, Apr 1998.

[Ste90] W R Stevens. *Unix Network Programming*. Prentice-Hall, 1990.

[Tan96] A S Tanenbaum. *Computer Networks*. Prentice-Hall, 1996.

[TVR$^+$99] W Townsley, A Valencia, A Rubens, G Paul, G Zorn, and B Palter. Layer Two Tunneling Protocol 'L2TP'. RFC 2661, Aug 1999.

[TW96] D L Tennenhouse and D J Weatherall. Towards an active network architecture. *Comp. Comm. Rev.*, 26(2), Apr 1996. see `http://www.tns.lcs.mit.edu/publications/ccr96.html`.

[Vah96] U Vahalia. *Unix Internals: the new frontiers*. Prentice-Hall, 1996.

[Wea99] D J Weatherall. *Service Introduction in Active Networks*. PhD thesis, MIT, Feb 1999.

[WHK97] M Wahl, T Howes, and S Kille. Lightweight Directory Access Protocol (v3). RFC 2251, Dec 1997.

[Yan03] X Yang. NIRA: A New Internet Routing Architecture. In *ACM FDNA 2003 Workshop*, Aug 2003. See http://www.isi.edu/newarch.

APPENDIX A

LIST OF ACRONYMS

AAL5 ATM adaptation layer 5, which defines *virtual paths* and *virtual circuits.*

ACL access control list. Term used in both networking and filesystems to specify restricted access, to subnets or hosts, and to directories or files, respectively. The format differs considerably, however, since in IP, one must list all possible address prefixes to (or from) which the access is to restricted.

AFS Andrew filesystem, a distributed filesystem that uses Kerberos for efficient, distributed authentication.

ALG application level gateway. See [CCR97].

ANTS Active Node Transfer System, an active networking concept. See [Wea99].

ANS Addressless Networking System. Refers to a "routing name space" constructed in this present thesis. Not to be confused for *ad hoc* or active network systems.

ANSI American National Standards Institute.

API application programming interface.

ARP address resolution protocol, which provides mapping of IP destination identifiers to their Layer 2 or LAN addresses.

AS autonomous system.

ATM asynchronous transfer mode. This and 802.6 are *Cell Relay* technologies, as opposed to *Frame Relay.*

AVES Address Virtualization Enablement Service. A scheme to effectively extend the IPv4 Internet address space locally, via external DNS name associations for server hosts behind a NAT firewall and connection relaying. See [NZS01].

bANS base ANS, an ANS with links configured using the infrastructure network addresses or methods, including TCP over IP infrastructure, a VC over ATM AAL5, or a VC over Sonet/FrameRelay/X.25, etc.

BGP Border Gateway Protocol. More specifically, BGP-4, the inter-domain routing protocol currently used in the Internet. See [RL95].

BOF birds-of-a-feather session, a requisite in the IETF for starting a workgroup.

CAN content addressible network. See [RFH$^+$01].

CCD charge-coupled device, a solid-state technology used in modern digital cameras and astronomical telescopes.

CCITT Consultative Committee for International Telegraph and Telephone.

CIO chief information officer, the person in a company that employees and network administrators alike love to keep in the dark about firewall holes and leaked routes, using the obfuscating power of IP! Not really, especially if you value your next year's equipment budget, but it happens all the time – see the white papers and case studies of Lumeta, at `http://www.lumeta.com`.

CLNP ConnectionLess Network Protocol, defined in ISO 8473.

CP canonical principle, more particularly, of networking, as described in this thesis.

dANS derivative ANS, using links formed by LRT entries of another ANS.

DDNS dynamic DNS.

DFS Distributed File System. A generic term that could be applied also to NFS. More particularly refers to an IBM Transarc product. See [Vah96].

DHCP Dynamic Host Control Protocol. The now *de facto* mechanism used by ISPs and intranets to configure (client) hosts. See [D$^+$97].

DHT distributed hash table. See [BLR$^+$04].

DNS Domain Name System. The *de facto* Internet name space. See [Moc87].

DTL designated transit list, list of switch ids representing a route in ATM, similar to source routing in IP.

EEA end to end arguments. A respected landmark in the evolution of the Internet philosophy, which called for separating issues like packet ordering and reliable delivery to the end host protocol stacks. Ref. [SRC84].

EEP end to end principle, essentially EEA.

FAIR Forward path setup with Acknowledgement, Inband signalling for Reverse path, the preferred LSP setup signalling protocol in the present architecture.

FARA Forwarding directive, Association and Rendezvous Architecture, see [CBFP03].

FQDN fully qualified domain names, of the DNS.

FTP file transfer protocol, which runs on top of TCP and is still very commonly used in the Internet.

GSM Groupe Speciale Mobile. The most widespread cellular telephone system.

GUI graphical user interface.

HIP host identity in the payload. See [Mos01].

HTTP hypertext transport protocol. The protocol used by the Web, and by virtually any application that wants to transport data over a firewall. See [GB01].

IAB Internet Architecture Board. A special group within the IETF that governs on architectural matters. See `http://www.iab.org`.

ICMP Internet Control Message Protocol, used notably by `ping` and `traceroute`. See [Pos81].

ICANN Internet Corporation for Assigned Names and Numbers, the international body currently responsible for coordinating generic (gTLD) and country-code (ccTLD) Top Level Domains of the DNS and the top level allocation of IP address blocks. This is ultimately what the present thesis aims to obviate.

IDP inter-domain protocol, as defined and provided by this thesis for the first time. Also, initial domain part in the ISO CLNP addressing scheme.

IEEE Institute of Electronics and Electrical Engineers. Responsible for a number of link level protocol standards, such as by the 802.1 Working Group.

IEN Internet Engineering Note. Name for a set of very early RFCs.

IETF Internet Engineering Task Force. See `http://www.ietf.org`.

INFS Internet FileSystem. A VFS for a network name space. Currently implemented only on Linux and only for the DNS, but an ANS implementation is intended sometime in the future. Has unique technical advantages even for DNS.

INS Intentional Networking System. See [AWSBL00].

IP Internet Protocol. In this thesis, mostly refers to the existing Internet addressing and routing architecture. Commonly used to denote the current Internet protocol suite in its entirety, including the *end point protocols* like TCP, UDP, etc., and other protocols used for routing or administration, like BGP, OSPF, ICMP, LDAP, etc.. This thesis shows that IP addressing and routing mechanisms would be better perceived as forming a general *intranet protocol* suite.

IPC inter-process communication, more specifically used for the scheme defined in Unix System V. See [Ste90].

IPNL IP Next Level, transitional architecture conceived as an interim alternative to IPv6, see [FG01].

IPv4 Internet Protocol version 4, the current Internet architecture (as of this thesis, in 2004), involving 32-bit addresses and lots of NAT.

IPv6 Internet Protocol version 6. With 128-bit addresses. Already showcased on the Internet2 and now supported on all major operating systems.

IRP Internet relay path, a multi-realm route in TRIAD.

IRTF Internet Research Task Force. See `http://www.irtf.org`.

ISO International Standards Organization, in English.

ISOC Internet Society, an international body that now oversees the global Internet.

ISP Internet Service Provider.

J2EE Java 2 platform, Enterprise Edition. See `http://java.sun.com/j2ee/`.

JIT just-in-time. Manufacturing technology term for minimization of inventory by scheduling parts to arrive only when actually needed for assembly.

KP Knowledge Plane, proposed by D Clark *et al.* as a next step to the present generation of network management protocols and software.

L2TP Layer 2 Tunneling Protocol. Tunnels Layer 2 traffic over IP. See [TVR$^+$99].

LAN Local Area Network. Refers to an Ethernet (or another) Layer 2 network in the OSI reference model, comprising a low level, local network address space of MAC addresses.

LDP label distribution protocol, also known as "control plane", in MPLS.

LDAP lightweight directory access protocol. See, for instance, [WHK97].

LRT lateral route table, specific to the present architecture. In principle, it comprises tuples of ANS node identifiers and the corresponding infrastructure addresses or link ids directly leading to those nodes.

LSP label switched path. A modern version of the X.25/Frame Relay/ATM virtual circuit (or path) routing concept, particularly used in MPLS.

MAC Media Access Control. Though they are commonly referred to as "addresses", they are unlike both postal and IP addresses, in that no part of a MAC address has any relation to routing. They are used as indices in Layer 2 route tables, and by the Ethernet interfaces to pick and choose packets coming their way.

MPI message passing interface. See [MPI97].

MPLS multiprotocol label switching. See [RVC01].

MX mail exchange SMTP records for redirecting email.

NAT network address translation. Scheme whereby designated blocks of addresses, e.g. the `10.0.0.0` subnet, is used by multiple client networks via address (and optionally port) translation mechanisms at the client network gateways, whereby all hosts within such a network are mapped to a single public Internet address belonging to its gateway. Often confused for a firewall because of its obscuring property. Similar to *bank switching* of memory, common in embedded systems, in which blocks of memory addresses are pointed to different storage areas.

NFS Network File System. Sun's (`http://www.sun.com`) networking of filesystems, widely used in the Unix world.

NIRA new Internet routing architecture. A recently proposed scalable and more robust alternative to BGP, which is strongly hierarchical and has an ideal fit into the present architecture [Yan03].

NSRG namespace research group in the IRTF, disbanded as of March 2004 without RFC.

OS operating system. Generally denotes host system software.

OSI Open Systems Interconnections. Reference model defined by ISO, mostly under the influence of SNA. Cited in numerous texts, including [Tan96,§1.4].

OSPF Open Shortest Path First. Link-state routing protocol now widely used as an interior routing protocol in edge and private networks. See [Hui00b].

OSS operating support system.

PC personal computer, as in a workstation, laptop, notebook or a PDA.

PDA personal digital assistant.

PNNI private network to network interface. This specification includes a detailed description of the native routing in ATM networks.

PVM parallel virtual machine. See [G$^+$94].

PVC permanent VC, mainly ATM terminology meaning manually configured VC.

QoS quality of service.

RFC request for comment. At one time, almost any working document in the IETF with presumably some consensus. Today, a working document that has graduated beyond the stage of an Internet-draft (which have limited lifespans), and possibly even a standard.

RISC reduced instruction set computing, a major paradigm in processor architecture.

RON Reliable Overlay Network. See [ABKM01].

RP representational principle. A tautological notion that the physical representational scheme for any given kind of knowledge is necessarily complete with respect to the representable knowledge.

RTP real time protocol.

RTSP real time streaming protocol, in reality, a control protocol for client-server sessions, with real time start-pause control operations.

SCTP stream control transmission protocol. See [OY02].

SDP Session Description Protocol, used with SIP, RTSP and related protocols. See [HJ98].

SIP Session Initiation Protocol. See [HSSR99].

SMS short message service, provided, for example, in GSM.

SMTP simple message transfer protocol, the Internet email system today.

SNA Systems Networking Architecture. IBM's mainframe networking architecture, now purely historical. Used synchronous link protocols.

SOAP Simple Object Access Protocol. See `http://www.w3c.org/TR/soap12`.

SPT Spanning Tree Algorithm. Famous bridge technique. See [Per99,p.58].

SRU subscribed route update, an LRT cache-invalidation/update notification mainly for derivative ANS links.

`ssh` secure shell. See `http://www.openssh.org`.

SSL secure sockets layer. See `http://www.openssl.org`.

SVC switched VC, mainly ATM terminology meaning a VC set up by signalling, and "soft" in this sense.

TCP transport control protocol, the basic reliable Layer 4 protocol in the current Internet. See also UDP and SCTP.

TLS transport layer security.

TRIAD Translating, Relaying Internetwork Architecture integrating Active Directories, a Stanford University project by Cheriton *et al.*. See [CG00].

UDP user datagram protocol, the basic unreliable Layer 4 protocol in the current Internet. See also TCP and SCTP.

URL universal resource locator, The standard form of the world wide web addresses.

UUCP Unix-to-Unix Copy, email relay scheme once widely used. Used sequences of locally configured hostnames to specify destinations.

VAS (asymmetric) virtual address space. Yet another coinage of the present thesis, to draw similarity to a basic feature in any modern OS, and to distinguish from

VC virtual connection or circuit.

VFS virtual file system. A generic interface within the OS for supporting a diverse variety of filesystems, such as NFS, DFS, etc. Also used to refer to any such instance that uses the VFS interface. See [Vah96].

VLAN Virtual LAN. A virtual Layer 2 address space, typically in a switched Ethernet environment or using L2TP [TVR+99]. Among other things, a VLAN would allow link-local traffic and IP route discovery protocols.

VoIP Voice-over-IP. Informal term for Internet telephony protocols and products (which may allow for low bandwidth video as well).

VPI virtual path index. In the present scheme, denotes an offset into a VPT.

VPN Virtual Private Network. A private IP address space mapping, typically over a secure network connection, to link to a remote host. This is Layer 3 as opposed to VLAN, which is at Layer 2.

VPT virtual path table, a table of LSP forwarding states.

WAN wide area network, denotes dedicated point-to-point links, typically over being widely used for structured application data over the Web.

WRAP widearea relay addressing protocol, part of TRIAD.

APPENDIX B

PAPERS, PATENTS AND DISCLOSURES

B.1 Summary of peer-reviewed papers

This research has produced workshop and conference presentations and also a journal paper over its five years of evolution. A summary of this progress is as follows:

J-1. ECUMN'2000: Basic name tree construction requiring link-local addresses only and its usability as a global address space for both datagram transport, as short message service (SMS) and connection set up.

This paper virtually dictated the opening plenary discussion by experts from European PTTs, as expressly acknowledged, who paraphrased the problem that even as Europe was rushing to IPv6 for "convergence", an analytical treatment of the addressing problem was lacking, and IP is unconvincing for this role.

J-2. NYNET'2001: Two papers were submitted and were both accepted. The main developments were (a) the virtual IP address pool, subsequently implemented in the prototype, and (b) the recursive aggregation protocol, described in more detail in the journal paper and fully in the patent disclosure (Appendix B.3).

J-3. INET'2001: This was only a poster presentation, but DNS query interception, virtual IP address pool and the connection set up signalling were completed and debugged by the third day of the conference, and successfully presented to Carpenter and others on an IBM 770Z Thinkpad running a Linux 2.4 kernel.

J-4. IWNA'2002: The quality of routing was the next major concern. One colleague (A. Joseph Hoane) had pointed out that the creation of the ANS nodes would be itself opportunistic, and hence that the tree routes could be as such fairly optimal by the very measures that presented the opportunities for deploying the ANS nodes. It was nevertheless desirable to seek a more general and assuring solution. The solution found in Fall 2001, presented at IWNA, is conventional route discovery adapted to the tree network.

J-5. Annales, July 2003 (invited 2001, revised: Sep 2002): The incremental progress reported in this paper is the unified treatment of tree (canonical) and discovered routes and routing of both datagrams and flows or connections.

This still left fundamental issues of naming, viz the insufficiency and inflexibility of a single name space, and of adequacy for routing on very large scales, to be addressed. Although the intuition of secondary ANS trees was being formulated at this time, I had no direct familiarity with provider networks (until joining Smarts in 2003), and lacked any authority of my own to justify the adequacy of routing. As explained, NIRA provided this justification a full year later in late 2003, permitting the first draft of this thesis by Spring 2004.

Of the three related patent disclosures reviewed and approved for filing at IBM, two had been filed by July 2002. Their abstracts are included below for reference. (The full specifications took 46 pages in the initial compilation of this thesis.)

B.2 US Patent 6,802,068 Addressless Internetworking

This was the first patent, filed as a continuation in 1999. The specification principally documents the routing and signalling later described in the ECUMN paper, and also a prior experimental system calls framework on AIX that was a precursor to the INFS scheme (Section 6.2).

Title US Patent 6,802,068: Addressless Internetworking

Particulars Filed: 3 Aug 1999. Issued: 5 Oct 2005. Assignee: IBM.

Abstract Uniform and infinitely scalable system and method for communication between application processes providing both point-to-point and multi-point connectivity without dependance on end-to-end addressing, using a framework of nameservers as exchanges for sharing named contexts of communication. Application processes define and reference the contexts by name on nameservers addressed by pathnames, and the framework then synthesises the end-to-end transport between the requesting processes by first concatenating the service paths taken by the defining and referencing requests to form end-to-end service paths, and then using these end-to-end service paths to perform the requisite signalling to the underlying physical networks for setting up the transport. Only local references are used in the configuration of the nameservers and switches, and in the computation and signalling of the service and transport paths, respectively. Each shared context provides a virtual network address space for multiple, simultaneous connections, and the contexts also serve as in-network framework for hosting connection management facilities, including network-level authentication, as well as transport mechanisms providing diverse qualities of service.

B.3 Filed application: Self-scaling network

This disclosure is a complete description of the recursive aggregation scheme described in the journal paper [Gur03] as a means for ensuring the scalability of connections or flows on very large scales. The scheme, in whole or in part, would be likely needed under the present approach as current alternatives depend strongly on the availability of end to end Layer 3 addresses, which would not be available.

This protocol would enable a connection/flow-oriented network to automatically recognize and exploit aggregation opportunities, making it possible, in the limit, to maintain the state volume at an individual switch close to a logarithm of the total number of connections passing through that switch. Even with a significant fraction of non canonical routes, this can be expected to compare favourably against the BGP route table sizes in the core networks of the Internet, as the latter depend on the size of the overall address space and granularity of advertisements. It is however difficult to make a more tangible comparison given the nature of differences.

Title Self-scaling network

Assignee IBM.

Abstract A label switched network, wherein parallel segments of the label switched paths are automatically identified and aggregated into tunnels, thus aggregating traffic within the core network, over and above aggregations that may be already performed end-to-end, and limiting the switch state volume to a logarithm of the number of end-to-end connections.

Parallel bundles of paths starting at a given switch are successively identified by an association signal tracing one or more of the paths, tunnels are then created along each of the bundles by an aggregation signal, and the paths comprising the bundles are then aggregated robustly and without packet loss in a finalisation pass, the process being repetitively initiated by multiple switches within the network. Paths, including other tunnels, running parallel to a given tunnel for a part of its way are aggregated with it in a shorter super-tunnel, and parallel tunnels are merged without additional nesting. Optionally, the tunnels may be extended by concatenation, one-way paths joining one-way bundles for the rest of their way also captured into the corresponding tunnels, and one-way tunnels extended and branched in the backward direction when newer associations are discovered ending at the start of existing one-way tunnels.